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

S. Arabas¹, A. Jaruga¹, H. Pawlowska¹, and W. W. Grabowski²

¹Institute of Geophysics, Faculty of Physics, University of Warsaw, Warsaw, Poland
²National Center for Atmospheric Research (NCAR), Boulder, CO, USA

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

Received: 18 August 2014 – Published in Geosci. Model Dev. Discuss.: 26 November 2014
Revised: 10 April 2015 – Accepted: 29 April 2015 – Published: 9 June 2015

Abstract. This paper introduces a library of algorithms for representing cloud microphysics 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 dimensionalities and complexities ranging from parcel models to multi-dimensional cloud-resolving (e.g. large-eddy) simulations. A two-dimensional (2-D) 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 one to leverage the power of graphics processing units (GPU), retaining the possibility of compiling the unchanged code for execution on single or multiple standard processors (CPUs). The paper includes a 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 the development of limited-area models as well (yet, such software-engineering issues are rarely the subject of discussion in the literature). As a consequence, there exist multiple implementations of the same algorithms, but it is difficult to dissect and attribute the differences between them. Avoiding “transplants” in the code is not easy, as numerous software projects in atmospheric modelling feature a 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). The 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.

The popularity of several geoscientific-modelling software packages that offer shared-library functionality suggests the

Published by Copernicus Publications on behalf of the European Geosciences Union.
soundness of such an approach – e.g. libRadtran (Mayer and Kylling, 2005) and CLUDBB (Golaz et al., 2002), cited nearly 350 and 100 times, respectively.

The motivation behind the development of the libcloudph++ 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 dimensionalities, different dynamical cores, different parallelisation strategies, and, in principle, models written in different programming languages. The 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 and 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 a 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 (e.g. Straka, 2009, 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 representa-

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

A 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 that allow one to use all of the libcloudph++ features from the Python programming language. The bindings are described in a separate technical note (Jarecka et al., 2015) which also includes 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 C contains a list of symbols used throughout the text. Appendix B 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 to 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-process 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 \( \rho_0(z) \) is used), which allows one to prescribe the 2-D velocity field using a stream function:

\[
\begin{align*}
\partial_t \rho_0 \cdot u &= -\partial_x \psi, \\
\partial_t \rho_0 \cdot w &= \partial_x \psi,
\end{align*}
\]

where \( \psi = \psi(x, z; t) \) is the stream function and \( u \) and \( w \) denote the horizontal and vertical components of the velocity field \( u \).

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, Muhl Bauer 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 stream function \( \psi(x, z) \) is the same as in Rasinski et al. (2011, Eq. 2):

\[
\psi(x, z) = -w_{\text{max}} \frac{X}{\pi} \sin \left( \frac{\pi z}{Z} \right) \cos \left( \frac{2\pi x}{X} \right),
\]

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

The initial profiles of liquid-water potential temperature \( \theta_l \) and the total water mixing ratio \( r_l \) are defined as constant with altitude (\( \theta_l = 289 \text{ K} ; r_l = 7.5 \text{ g kg}^{-1} \)). The initial air-density profile corresponds to the hydrostatic equilibrium with a pressure of 1015 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:

\[
N(r_d) = \sum_m \frac{N_m}{\sqrt{2\pi \ln(\sigma_m)}} \frac{1}{r_d} \exp \left[ -\left( \frac{\ln(r_d)}{\sqrt{2\ln(\sigma_m)}} \right)^2 \right],
\]

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

\[
\begin{align*}
\sigma_1 &= 1.4; \quad d_1 = 0.04 \mu\text{m}; \quad N_1 = 60 \text{ cm}^{-3}, \\
\sigma_2 &= 1.6; \quad d_2 = 0.15 \mu\text{m}; \quad N_2 = 40 \text{ cm}^{-3},
\end{align*}
\]

where \( \sigma \) is the geometric standard deviation, \( d_{1,2} \) is the mode diameter and \( N_{1,2} \) is the particle concentration.
under standard conditions \((T = 20^\circ C\) and \(p = 1013.25\ hPa)\).
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 the vertical gradient of relative humidity. Both modes of the distribution are assumed to be composed of ammonium sulfate.

In the examples presented in this paper, the model was initialised with \(\theta = \theta_1\) and \(r_v = r_i\) (i.e. no condensed water). To avoid unrealistic supersaturation, an arbitrary limit of 5\% (RH = 1.05) was imposed when evaluating the 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 \(\phi_0\), \(\langle \phi \rangle\) and \(\tau\) are the initial profile, the horizontal mean of \(\phi\) at a given height and the relaxation timescale, respectively. The relaxation timescale \(\tau\) is height-dependent (mimicking effects of surface heat fluxes) and is prescribed as \(\tau = r_{\text{fck}} \cdot \exp(z/z_{\text{fck}})\) with \(r_{\text{fck}} = 300 s\) and \(z_{\text{fck}} = 200 m\). Note that such a formulation does not dampen small-scale perturbations of \(\phi\), but simply shifts the horizontal mean toward \(\phi_0\).

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 2 h 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 time step is 1 s. Shorter sub-time-steps 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

\[
\partial_t r_i + \frac{1}{\rho_d} \nabla \cdot (u \rho_d r_i) = \dot{r}_i, \tag{4}
\]

where \(r_i\) is the mixing ratio of the advected constituent, \(\rho_d\) is the dry-air density, \(u\) 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 time step. 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 time steps.

The proposed solver features uncentered-in-time integration of the right-hand-side terms. The source terms that are 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
Table 1. State variables for the three implemented schemes. Number of state variables times the number of Eulerian grid cells plus the number of particle attributes times the number of Lagrangian computational particles gives an estimation of the memory requirement of a given scheme. See Appendix C for symbol definitions.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>State variables</th>
<th>Particle attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eulerian (PDE)</td>
<td>$\theta, r_v, r_c, r_f$</td>
<td>–</td>
</tr>
<tr>
<td>Lagrangian (ODE)</td>
<td>$\theta, r_v, r_c, r_f, n_c, n_t$</td>
<td>–</td>
</tr>
<tr>
<td>Particle-based</td>
<td>$\theta, r_v$</td>
<td>$r_d^3, r_q^2, N, \kappa$</td>
</tr>
</tbody>
</table>

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 $\theta$ 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 to 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 parameterized 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 one 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:

$$\Delta \theta = \int \frac{d\theta}{r_c} dr_v + \int \frac{d\theta}{r_r} dr_v$$

$$\Delta r_v = \left( r_v^p - r_v \right) + \left( r_v^c - r_v \right),$$

where $\frac{d\theta}{dr_v} = \frac{-\rho_l}{v_s \theta_l} \Delta r_v$ (cf. Eq. A13 in Appendix A) and the integration limit $r_v^p$ for cloud water condensation/evaporation is

$$r_v^p = \begin{cases} r_v^s & r_v > r_v^s \\ r_v^s & r_v \leq r_v^s \wedge r_v \leq r_v^c - r_v^s - r_v \\ r_v^c + r_v & r_v \leq r_v^s \wedge r_v < r_v^c - r_v^s - r_v, \end{cases}$$

where $r_v^s = r_v^s(\rho_0, \theta, r_v^p)$ 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 sub-saturation 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 $r_v^c$ defined as follows:

$$r_v^c = \begin{cases} r_v^c \quad r_v^c > r_v \\ r_v^c \quad r_v < r_v^c \wedge \delta r_t \geq r_v^c - r_v \\ r_v^c + \delta r_t \quad r_v < r_v^c \wedge \delta r_t < r_v^c - r_v, \end{cases}$$

where $\delta r_t$ depicts the limit of evaporation of rain within one time step. Here, it is parameterised as $\delta r_t = \min(r_t, \Delta t \cdot E_t)$, with $E_t$ 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 $r_v^p$, here $r_v^c = r_v^c(\rho_0, \theta, r_v^c)$.

Noteworthy: the name adjustment reserved in Sect. 2.3 for source terms not formulated as time derivatives suits the above-defined procedure, which is formulated through integration over the vapour mixing ratio and not over time.
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 formulae. See Wood (2005) for a review of how these formulations compare with other bulk warm-rain schemes.

In the Kessler formulation, the autoconversion source term is proportional to $\max(r_c - r_c0, 0)$, 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 $10^{-3}$ 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:

\[
\begin{align*}
\dot{r}_t^{\text{new}} &= \dot{r}_t^{\text{old}} - \frac{(F_{\text{in}} - F_{\text{out}})}{\rho_d} \\
F_{\text{in}} &= F_{\text{out}}|_{\text{above}} \\
F_{\text{out}} &= -\frac{\dot{r}_t}{\Delta z} \left[ \rho_d v_1^{\text{below}}(\dot{r}_t|_{\text{below}}) + \rho_d v_1(\dot{r}_t) \right],
\end{align*}
\]

where $^{\text{old}}$ and $^{\text{new}}$ superscripts are introduced to indicate that $\dot{r}_t$ is a sum of multiple terms. The $|_{\text{above}}$ and $|_{\text{below}}$ symbols refer to the grid cell sequence in a column, $v_1$ is the rain terminal velocity parameterised as a function of rain water mixing ratio (Eq. (5d) in Grabowski and Smolarkiewicz, 1996), and $F_{\text{in}}$ and $F_{\text{out}}$ symbolise fluxes of $r_t$ through the grid cell edges.

Employment of the upstream scheme brings several consequences. First, unlike the cell-wise 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 $\Delta z$ and time step $\Delta t$ 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 (e.g. MPDATA, cf. Smolarkiewicz, 2006, and references therein).

---

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

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. cell-wise or column-wise calculations). In the case of the single-moment bulk scheme, the three functions correspond to the three represented processes, namely phase changes (cell-wise adjustments), coalescence (cell-wise rhs terms), and sedimentation (column-wise rhs term). Sedimentation is the only process involving column-wise traversal of the domain (note the $|_{\text{above}}$ and $|_{\text{below}}$ 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 the floating point format (e.g. float, double, etc.). 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, $r_c0 = 5 \times 10^{-4}$ kg kg$^{-1}$ and the tolerance is set to $2 \times 10^{-5}$ 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).

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 B).

Listing 1. blk_1m::opts_t definition.

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

$^2$C++ Standard Template Library
template<typename real_t>
struct opts_t {
bool
cond = true, // condensation
cevp = true, // evaporation of rain
revp = true, // evaporation of cloud
sedi = true; // sedimentation
real_t
r_c0 = 5e-4, // autoconv. threshold
r_eps = 2e-5; // absolute tolerance
};

The function arguments include references to containers storing \( \rho_0 \) (read-only) and \( \theta, r_v, r_c, r_f \) (to be adjusted). The last argument \( dt \) is the time step needed to calculate the precipitation evaporation limit (see the discussion of Eq. 8). Forcings due to autoconversion and accretion are obtained through a call to the \texttt{blk\_1m::rhs\_cellwise()} function whose signature is given in Listing 3. The function modifies \( r_v \) and \( r_c \) by adding the computed rhs terms to the values already present in \( r_v \) and \( r_c \). Read-only access is required for \( \rho_0, r_v \) and \( r_c \) passed as the last three arguments.

Representation of sedimentation is included in a separate function \texttt{rhs\_columnwise()} (signature in Listing 4). The \texttt{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.

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

The integrals in Eq. (7) defining the saturation adjustment procedure are computed using the Boost.Odeint library (Ahrent 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 B 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.
Figure 3. Sequence diagram of libcloudph++ API calls for the single-moment bulk scheme and a prototype transport equation solver. See the discussion in Sect. 3.2.2 and the caption of Fig. 2 for a description or the diagram elements.

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.
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 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, cloud water mixing ratio, rain drop concentration, and rain water mixing ratio 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 in 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 (log-normal, 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 by 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 one 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 log-normal size distribution of aerosols. Freshly activated cloud droplets are assumed to have a radius of 1 µm; 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 in aerosol concentration due to previous activation, in each time step the number of available aerosols is approximated by the difference between the initial aerosol concentration and the concentration of pre-existing 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 the 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 an initial radius of 25 µm.

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 $\rho_{d0}/\rho_d$ to follow Eq. (A4) in Morrison et al. (2005), where $\rho_{d0}$ is the density of dry air under 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 log-normal size distribution (see Eq. 3) and the parameter $\beta$ 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, the 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 added to the values already present in the arrays passed as arguments.

The cell-wise-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 these processes, multiplied by the time step, will be 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 an argument using the following rule:

$$\dot{r}_i^\text{new} = \min\left(\dot{r}_i^*, \dot{r}_i^\text{old} + \Delta t \cdot \dot{r}_i^\text{old} / \Delta t\right).$$  (12)

where $\dot{r}_i^\text{old}$ is the value obtained in the evaluation of previously-handled processes, $\dot{r}_i^*$ is the value computed using the model formulae, and $\dot{r}_i^\text{new}$ is the augmented value of the rhs term that guarantees non-negative values of $r_i$ after its application. The same rule is applied when evaluating values of outgoing fluxes $F_{\text{out}}$ from Eq. (9) when calculating rhs term within rhs_columnwise(). The rhs_columnwise() returns the value of the $F_{\text{out}}$ 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
Figure 5. Sequence diagram of libcloudph++ API calls for the double-moment bulk scheme and a prototype transport equation solver. See the discussion in Sect. 4.2.2 and the caption of Fig. 2 for a description of the diagram elements.

difference from the single-moment bulk scheme’s calling sequence presented in Sect. 3.2.2 is the lack of an adjustments 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 is 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 concentration 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 $\theta$ 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 zero, one, two or three components),
- wet radius squared $r^2_w$,
- dry radius cubed $r^3_d$, and
- hygroscopicity parameter $\kappa$.

Multiplicity depicts the number of particles represented by a computational particle. All particles represented by one computational particle are assumed to be spherical aqueous solution droplets of radius $r_w$. Following Shima et al. (2009), the model is formulated in $r^2_w$ for numerical reasons.
The amount of solvent is represented by the dry radius \( r_d \) (the third power is used in the model code because most often \( r_d^3 \) serves as a proxy for the 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 composed of an aqueous solution and spherically shaped. 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 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 breakups 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 will pass 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 time step 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:

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

where \( C \) is the Courant number field component, and \( \Delta x \) is the grid step (formulae are applicable for 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):

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

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

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

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

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

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):

\[
r_w \frac{dr_w}{dt} = \frac{D_{\text{eff}}}{\rho_w} (\rho_v - \rho_s),
\]

where the effective diffusion coefficient is

\[
D_{\text{eff}}^{-1} = D^{-1} + K^{-1} \rho_v \frac{v_r}{T} \left( \frac{l_v}{R_v T} - 1 \right)
\]

and \( \rho_v \) stands for the density of water vapour at saturation with respect to a plane surface of pure water. The vapour density at drop surface \( \rho_0 \) is modelled as

\[
\rho_0 = \rho_v \cdot a_w (r_w, r_d) \cdot \exp(A/r_w),
\]

where

\[
\begin{align*}
A &= 
\end{align*}
\]
where water activity \( a_w \) and the so-called Kelvin term \( \exp(A/r_w) \) are evaluated using the \( \kappa \)-Köhler parameterisation of Petters and Kreidenweis (2007). See Arabas and Pawlowska (2011) for the formulae for \( A \), \( l_\gamma \) and \( \rho_w \).

Vapour and heat diffusion coefficients \( D \) and \( K \) are evaluated as

\[
D = D_0 \cdot \beta_M \cdot \frac{Sh}{2} \quad (20)
\]

\[
K = K_0 \cdot \beta_T \cdot \frac{Nu}{2} \quad (21)
\]

The Fuchs–Sutugin transition-régime correction factors \( \beta_M(r_w, T) \) and \( \beta_T(r_w, T, p) \) 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 Sect. 5.1.5 herein).

The representation of condensation and evaporation in the Lagrangian component involves a sub-stepping logic in which the Eulerian component time step \( \Delta t \) is divided into a number of equal sub-steps. This is intended to cope with potentially large differences 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 sub-time-steps is kept constant throughout the domain and throughout the simulation time. However, the actual constraints for time-step length \( \Delta t' \) differ substantially, particularly with the distance from cloud base (see Fig. 2 in Arabas and Pawlowska, 2011). An adaptive time-step choice mechanism is planned for a future release. For simplicity, the sub-stepping procedure is not depicted explicitly in the following formulae. It is only hinted at by labelling the sub-time-step as \( \Delta t' \) and the sub-time-step number as \( n' \). If the user enables sub-stepping, the advective tendencies of \( \theta_d \) 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 the wet radius but explicit with respect to \( r_v \) and \( \theta \):

\[
r_w^{n' + 1} = r_w^{n'} + \Delta t' \cdot \frac{dr_w}{d\tau} |_{r_w^{n'}, \beta_w^{n'}} \quad (22)
\]

The solution to the above equation is sought by employing a predictor–corrector type procedure. First, the value of the \( \frac{dr_w}{d\tau} \) derivative evaluated at \( r_w^{n'} \) is used to construct an initial-guess range \( a < r_w^{n' + 1} < b \) in which roots of Eq. (22) are to be sought, with

\[
a = \max \left( r_w^{2n'} + \min \left( 2 \cdot \frac{dr_w}{d\tau} |_{r_w^{n'}}, 0 \right) \right) \quad (23)
\]

\[
b = r_w^{2n'} + \max \left( 2 \cdot \frac{dr_w}{d\tau} |_{r_w^{n'}}, 0 \right) \quad (24)
\]

Second, \( r_w^{n' + 1} \) is iteratively searched using the bisection algorithm. If the initial-guess range choice makes a bisection search ill-posed (the minimisation function having the same sign at \( a \) and \( b \)), the algorithm stops after the first iteration, returning \((a + b)/2\), which 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 the 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 \( \theta \) are adjusted to account for water vapour content change due to condensation or evaporation on particles within a given grid cell and with a given sub-step by evaluating

\[
1 \sum_{i \in \text{grid cell}} N[i] \left( \frac{r_w^{3[n']^i + 1} - r_w^{3[n']^i}}{3 \Delta V} \right) \quad (25)
\]

\[
\delta[n' + 1] - \delta[n'] = \left( r_v^{n' + 1} - r_v^{n'} \right) \frac{d\theta}{dr_v} |_{r_v^{n'}, \theta[n']} \quad (26)
\]

where \( \Delta V \) is the grid cell volume, and \( \rho_w \) is the density of liquid water. Noteworthy: such a formulation maintains conservation of heat and moisture in the domain regardless of the accuracy of the integration of the drop growth equation.

Phase-change calculations are performed before any other processes. This is because condensation and evaporation are the only processes modifying the \( r_v \) and \( \theta \) fields of the Eulerian component. Consequently, the 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 is done for phase changes, coalescence of particles is solved using sub-time-steps $\Delta t''$. In each sub-time-step, 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

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

(27)

where $n$ is the total number of computational particles within a grid cell in a given time step 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 $n/2$ 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:

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

(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 by 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 a 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 sub-time-step, 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 (see Fig. 1 and Sect. 4.1.4 in Shima et al., 2009). 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 particle. 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 $\kappa$).

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 time step. The recycling procedure first looks for computational particles with the 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 time-step 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 Sect. 5.1.4 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 (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 a cell-wise 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 the radius. The minimal and maximal values of the dry radius are chosen automatically by evaluating the initial dry-size distribution. The criterion is that the particle multiplicity (i.e. the
threshold relative humidity (e.g. RH ≤ 50 %) is set to its equilibrium value at a given main. The initial wet radius of particles within the super-saturated regions is set to zero. The RH limit for evaluating the drop growth equation (e.g. during a spin-up period; see Sect. 2.2).

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 control over process toggling. The RH_max field defines the RH limit for evaluating the 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 functions returning the concentration of particles at STP as a function of logarithm of dry radius. Subsequent fields specify the geometry of the Eulerian grid and the time step. 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, and 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 time step when calculating condensation and coalescence is defined by sttp_cond and sttp_coal, respectively. The last two fields provide control of the initial mean concentration of computational particles per grid cell and the type of coalescence kernel to be used. As of the current release, two options are available: the geometric kernel and the Golovin kernel; see Listing 10).

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 meanings of the dataZero and strides fields match those of equally named methods of the Blitz++ Array class. Quoting the Blitz++ documentation (Veldhuizen, 2005): "dataZero is a pointer to the element (0, 0, ..., 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, which allows the library 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
enum kernel_t { geometric, golovin };

Listing 8. lgrngn::opts_t definition.

```cpp
template<typename real_t>
struct opts_t
{
    // process toggling
    bool adve, sedi, cond, coal;
    // RH limit for drop growth
    real_t RH_max;
}
```

Listing 10. lgrngn::kernel_t definition.

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

Listing 11. lgrngn::arrinfo_t definition.

```
Listing 9. lgrngn::opts_init_t definition.

```cpp
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 ststp_cond, ststp_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;
}
```

instance of the lgrngn::particles_t class (see Listing 12). Internally, the Lagrangian calculations are implemented using the Thrust library\(^3\), which, among other things, allows one 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 the 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>

\(^3\)http://thrust.github.io/
\(^4\)http://openmp.org/
\(^5\)http://nvidia.com/
The \texttt{particles\_t}'s methods prefixed with \texttt{diag\_} provide a mechanism for obtaining statistical information on the droplet parameters gridded on the Eulerian component mesh. The \texttt{particles\_t::diag\_sd\_conc()} method calculates the concentration of computational particles per cell. The \texttt{particles\_t::diag\_dry\_mom()} and \texttt{particles\_t::diag\_wet\_mom()} calculate statistical moments of the dry and wet size spectra respectively. The $k$th moment

\begin{equation}
M_{d,w}^{[k]} = (\rho_d \Delta V)^{-1} \sum_{r\in \text{grid cell}} N_{i}[i] d_{d,w}[i]^k,
\end{equation}

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 $[r_{mi}, r_{mx}]$ over which the moments are calculated is chosen by calling \texttt{diag\_dry\_rng()} or \texttt{diag\_wet\_rng()} before calls to \texttt{diag\_dry\_mom()} and \texttt{diag\_wet\_mom()}, respectively. The \texttt{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 \texttt{factory()} function. It returns a pointer to a newly allocated instance of the \texttt{particles\_t} class. Its arguments are the backend type (see Listing 13) and the scheme’s options grouped in the \texttt{opts\_init\_t} structure (see Listing 9). The purpose of introducing the \texttt{lggrngn::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 \texttt{particles\_t}). Second, it does report an error if the library was compiled without CUDA (GPU) or OpenMP (multi-threading) backend support.

\subsection{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 were 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 time step, for example, to provide data on a particle
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.
surface for a radiative-transfer component, or the data on a 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 one 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); and

- the possibility of performing part of the microphysics computations asynchronously, simultaneously with other computations carried out on CPU(s) (cf. Sect. 5.1.3).

Since the version of the 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 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 when 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 B), it is implemented using the C++11’s `std::async()` call.

Both in the cases of GPU and CPU configurations, the Mersenne Twister (Matsumoto and Nishimura, 1998) random number generator is used. When using GPU, the CUDA cuRAND’s MTGP32 is used. Offering parallel execution with multiple random number streams. When not using GPU, the C++11 `std::mt19937` is used and the random number generation is done by a single thread only, even when 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 Sect. 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 2 h long spin-up period). The initial mean concentration of computational particles was set to 64 per cell. The number of sub-steps was set to 10 for both condensation and coalescence. The geometric coalescence kernel was used, with the collection efficiency arbitrarily set to 0.5. With the employed simulation set-up, this causes the timing of precipitation onset to roughly match the one of the double-moment scheme.

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 $r_c$, $r_i$ (cf. Figs. 4 and 6) and $n_c$ and $n_i$ (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 μm 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 500 m above the bottom boundary of the domain in both cases.

The plot of the effective radius in Fig. 8 shows the gradual increase in 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
Figure 8. Example results from a 2-D kinematic simulation using the particle-based scheme. All panels depict the 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 a 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 a discussion.
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 the anticipated presence of the interstitial aerosol within the cloud. The regions of the 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.

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 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 B. 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, and
- all of the 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 -O3, -march = native and -DNDEBUG options enabled. The GPU code was compiled with nvcc 6.0 with -O3, -use_fast_math 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 time steps in the first run and a dozen time steps in the second run. The long and short run times were subtracted and the result was normalised by the difference in the number of time steps.

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 two threads,
- OpenMP backend using four threads, and
- CUDA backend using the GPU.

The test was completed for single-precision arithmetics. The GPU used offered about 3 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 as long as the single-moment scheme to advance the solution by one time step. The particle-based scheme may be anything from about 10 to over 100 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 sub-time-steps used for phase changes and coalescence (not shown; 10 sub-time-steps per one advective step were used here). 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 five- to ten-fold 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 the 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 in 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 done with maintainability and auditability as priorities. This is reflected in

- the choice of C++ with its concise and modularity-encouraging syntax;
- the separation of code elements related to the schemes’ formulation (formulae) 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 a substantial part of the library implementation to external libraries (including the dimensional analysis, algorithm parallelisation and GPU hardware handling); and
- the hosting of library development and handling of code dissemination through a public code repository.

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

---

As of the 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 pertain to the single-moment, double-moment, and particle-based schemes, respectively; ca. 1000 LOC define the Python bindings.
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 of 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 \( \rho_i \) and mass mixing ratios \( r_i \) are defined as follows:

\[
\begin{align*}
\rho_d & \quad \text{dry air} \\
\rho_v & = r_v \rho_d & \text{water vapour} \\
\rho_l & = r_l \rho_d & \text{liquid water}
\end{align*}
\]

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):

\[
\begin{align*}
\partial_t (\rho_d \dot{r}_v) + \nabla \cdot (\rho_d \dot{u} \dot{r}_v) &= 0 \\
\frac{D \dot{r}_v}{Dt} &= \dot{r}_v & \quad \text{(A4)} \\
\frac{Ds}{Dt} &= \dot{q} & \quad \text{(A5)}
\end{align*}
\]

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 \( 1 \gg r_v > r_l \).

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 \( \rho_v \) is obtained by summing Eq. (A4) - \( \rho_d + r_v \cdot \dot{r}_v \). Eq. (A3):

\[
\partial_t (\rho_v \dot{r}_v) + \nabla \cdot (\rho_v \dot{u} \dot{r}_v) = \dot{r}_v
\]

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

\[
ds = c_p^* \, d(\ln \theta^*)
\]

gives

\[
c_p^* \frac{d\theta^*}{dr} = \frac{\theta^*}{T} \dot{q}.
\]

At this point, no assumption is made about the exact form of \( \theta^* \) or \( c_p^* \). Summing Eq. (A3) - \( \theta^* \rho_v \) and Eq. (A8) - \( \dot{r}_v \) and \( \dot{q} \).

\[
\frac{D}{Dt} c_p^* \rho_v \theta^* = \dot{r}_v \theta^* \rho_v \dot{q} / T \quad \text{results in a continuity equation for } \rho_v c_p^* \theta^* \quad \text{(A9)}
\]

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

\[
\partial_t (\rho \phi) + \nabla \cdot (\rho \dot{u} \phi) = \rho \dot{q} \phi
\]

representing transport of a quantity \( \phi \) (equal to \( r_v \) or \( c_p^* \theta^* \)) 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 \( \theta^* \) and \( \dot{q} \). For moist air containing suspended water aerosol, assuming thermodynamic equilibrium and neglecting the expansion work of liquid water, \( ds \) may be expressed as (Eqs. 6.10 and 6.11 in Curry and Webster, 1999)

\[
ds = c_{pd}(ln \theta) \\
\begin{align*}
&= \frac{\dot{q}}{\dot{q}} - \frac{\dot{r}_v}{\dot{r}_v} + \frac{\dot{r}_v}{\dot{r}_v} + \frac{\dot{r}_v}{\dot{r}_v} / T
\end{align*}
\]

where \( \rho_d = \rho_d R_d T \) is the partial pressure of dry air, and the potential temperature \( \theta \) is defined here as

\[
\theta = T \left( \frac{p_{1000}}{p_d} \right) c_{pd}(ln \theta)
\]

\( (p_{1000} = 1000hPa; \) 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 \( c_{pd}^* = c_{pd} = \text{const} \) and \( \theta^* = \theta \) into Eq. (A9) and employing the form of \( \dot{q} \) hinted with \( -dq \) in Eq. (A11) gives

\[
\partial_t (\rho \dot{q}) + \nabla \cdot (\rho \dot{u} \dot{q}) = \frac{-\rho \dot{q} \dot{r}_v}{c_{pd} T} - \frac{\dot{r}_v \dot{r}_v + \dot{r}_v \dot{r}_v + \dot{r}_v / T}{\dot{r}_v / T}
\]

Neglecting all but the \( \dot{r}_v \) 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 \( \theta^* \) and \( \dot{q} \) is obtained by putting \( \theta^* = \theta \exp \left( \frac{c_{pd}(ln \theta^*)}{c_{pd}^*} \right) \), which results in the \( \dot{r}_v \) term becoming a part of \( c_{pd}(ln \theta^*) \) instead of \( -dq \) 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 $r_v$ linked to 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, and

$\theta$ potential temperature.

Assuming $\rho_d$ is known (solved by a dynamical core of a model), temperature and pressure may be diagnosed from $r_v$ and $\theta$ with

$$T = \left[ \theta \left( \frac{\rho_d R_d}{P_{1000}} \right)^{\frac{\rho_d}{\rho_d - R_d}} \right] \frac{\Theta}{\rho_d^c_p (\rho_d - R_d)}$$  \hspace{1cm} (A14)

$$p = \rho_d (R_d + r_v R_v) T. \hspace{1cm} (A15)$$
Appendix B: 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\(^7\) collection. The libmpdata++ components solve the transport equations for the Eulerian fields using the MPDATA algorithm (Smolarkiewicz, 2006) and provide a data output facility using the HDF5 library\(^8\). Figure B1 presents a 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\(^9\) 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\(^10\). Compilation instructions are provided in the README file.

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

\>$\text{icicle}\ -\text{help}$

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:

\>$\text{icicle}\ -\text{micro}\text{-lgrngn}\ -\text{help}$.

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 time step, 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.

Figure B1. A tree of libcloudph++’s and icicle’s major dependencies. In addition to these libraries, several components require the C++11 compiler and CMake at build time.

\(^7\)http://boost.org/
\(^8\)http://hdfgroup.org/
\(^9\)http://debian.org/
\(^10\)http://cmake.org/
### Appendix C: List of symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>SI unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A = 2a_w/(R_T r_w)$</td>
<td>(m)</td>
<td>Kelvin term exponent parameter</td>
</tr>
<tr>
<td>$\beta_M, \beta_T$</td>
<td>(1)</td>
<td>Transition régime correction factors</td>
</tr>
<tr>
<td>$\Delta t, \Delta x, \Delta z, \Delta V$</td>
<td>(s) or (m) or (m$^3$)</td>
<td>Time step, grid cell dimensions and volume</td>
</tr>
<tr>
<td>$\theta_i$</td>
<td>(K)</td>
<td>Liquid water potential temperature (cf. Sect. 2.2)</td>
</tr>
<tr>
<td>$\kappa$</td>
<td>(1)</td>
<td>Hygroscopicity parameter</td>
</tr>
<tr>
<td>$\rho_i$</td>
<td>Depends on $i$</td>
<td>Any state variable (density)</td>
</tr>
<tr>
<td>$\rho_0$, $\rho_v$</td>
<td>(kg m$^{-3}$)</td>
<td>Densities of dry air and vapour vapour</td>
</tr>
<tr>
<td>$\rho_c$, $\rho_r$</td>
<td>(kg m$^{-3}$)</td>
<td>Cloud and rain water densities/content</td>
</tr>
<tr>
<td>$\rho_w = 1000$</td>
<td>(kg m$^{-3}$)</td>
<td>Density of liquid water</td>
</tr>
<tr>
<td>$\rho_{ws}$</td>
<td>(kg m$^{-3}$)</td>
<td>Saturation vapour density</td>
</tr>
<tr>
<td>$\rho_w$</td>
<td>(kg m$^{-3}$)</td>
<td>Vapour density at drop surface</td>
</tr>
<tr>
<td>$\rho_l$, $\rho_r$, $\rho_t$</td>
<td>Depends on $i$</td>
<td>rhs terms (any, cloud water, rain water)</td>
</tr>
<tr>
<td>$\sigma_{m}$</td>
<td>(1)</td>
<td>Geometric standard deviation (log-normal spectrum)</td>
</tr>
<tr>
<td>$\sigma_{w} = 0.072$</td>
<td>(J m$^{-2}$)</td>
<td>Surface tension coefficient of water</td>
</tr>
<tr>
<td>$\tau$, $\tau_{dx}$</td>
<td>(s)</td>
<td>Relaxation timescale (cf. Sect. 2.2)</td>
</tr>
<tr>
<td>$\phi_i$</td>
<td>Depends on $i$</td>
<td>particle concentration (log-normal spectrum)</td>
</tr>
<tr>
<td>$\psi$</td>
<td>(kg m$^{-3}$ s$^{-1}$)</td>
<td>Any advected specific quantity (e.g. mixing ratio)</td>
</tr>
<tr>
<td>$\alpha_w = (r_0^3 - r_3^3)/(r_0^3 - r_d^3 \cdot (1 - \kappa))$</td>
<td>(1)</td>
<td>Stream function</td>
</tr>
<tr>
<td>$\alpha$, $b$</td>
<td>(m$^2$)</td>
<td>Initial interval for bisection algorithm</td>
</tr>
<tr>
<td>$c_{pd} = 1005$, $c_{pv} = 1850$, $c_1 = 4218$</td>
<td>(J kg$^{-1}$ K$^{-1}$)</td>
<td>Specific heat at const. pressure (dry air, vapour and liquid water)</td>
</tr>
<tr>
<td>$C$</td>
<td>(1)</td>
<td>Courant number</td>
</tr>
<tr>
<td>$d_m$, $r_m$</td>
<td>(m)</td>
<td>Mode diameter and radius (log-normal spectrum)</td>
</tr>
<tr>
<td>$D$, $D_{diff}$, $D_0$</td>
<td>(m$^2$ s$^{-1}$)</td>
<td>Diffusion coefficients for water vapour in air</td>
</tr>
<tr>
<td>$E_i$</td>
<td>(kg m$^{-3}$ s$^{-1}$)</td>
<td>Evaporation rate of rain (single-moment scheme)</td>
</tr>
<tr>
<td>$E(r_i, r_j)$</td>
<td>(1)</td>
<td>Collection efficiency</td>
</tr>
<tr>
<td>$F_{in}$, $F_{out}$</td>
<td>(kg m$^{-3}$ s$^{-1}$)</td>
<td>Fluxes of $\rho_i$ through the grid cell edges</td>
</tr>
<tr>
<td>$K$, $K_0$</td>
<td>(J m$^{-1}$ s$^{-1}$ K$^{-1}$)</td>
<td>Thermal conductivities of air</td>
</tr>
<tr>
<td>$K(r_i, r_j)$</td>
<td>(m$^3$ s$^{-1}$)</td>
<td>Collection kernel</td>
</tr>
<tr>
<td>$l_{0} = 2.5 \times 10^6$</td>
<td>(J kg$^{-1}$)</td>
<td>Latent heat of evaporation at the triple point</td>
</tr>
<tr>
<td>$l_k(T) = l_0 + (c_{pv} - c_j) \cdot (T - T_0)$</td>
<td>(J kg$^{-1}$)</td>
<td>Latent heat of evaporation at a given temperature</td>
</tr>
<tr>
<td>$M_{[k]}$</td>
<td>(m$^{3+k}$)</td>
<td>$k$th moment of size spectrum</td>
</tr>
<tr>
<td>$n$</td>
<td>(1)</td>
<td>Total number of computational particles</td>
</tr>
<tr>
<td>$n_c$, $n_r$</td>
<td>(m$^{-3}$)</td>
<td>Cloud droplet and rain drop concentrations</td>
</tr>
<tr>
<td>$N$</td>
<td>(1)</td>
<td>Multiplicity (attribute of computational particle)</td>
</tr>
<tr>
<td>$N_m$</td>
<td>(m$^{-3}$)</td>
<td>Particle concentration (log-normal spectrum)</td>
</tr>
<tr>
<td>$P$, $P_d$</td>
<td>(Pa)</td>
<td>Pressure, dry air partial pressure</td>
</tr>
<tr>
<td>$P_{ij}$</td>
<td>(1)</td>
<td>Probability of collisions</td>
</tr>
<tr>
<td>$Q$, $q$</td>
<td>(J m$^{-3}$), (J kg$^{-1}$)</td>
<td>Heat per unit volume and mass</td>
</tr>
<tr>
<td>$r_{d}$, $r_{w}$</td>
<td>(m)</td>
<td>Particle dry and wet radii</td>
</tr>
<tr>
<td>$r_{c}$</td>
<td>(kg kg$^{-1}$)</td>
<td>Autoconversion threshold (mixing ratio)</td>
</tr>
<tr>
<td>$r_{v}$, $r_{r}$, $r_{i} = r_{v} + r_{r}$</td>
<td>(kg kg$^{-1}$)</td>
<td>Mixing ratios (vapour, liquid, total)</td>
</tr>
<tr>
<td>$R_v$, $R_d$</td>
<td>(J K$^{-1}$ kg$^{-1}$)</td>
<td>Gas constants for water vapour and dry air</td>
</tr>
<tr>
<td>$S$, $s$</td>
<td>(J K$^{-1}$ m$^{-3}$), (J K$^{-1}$ kg$^{-1}$)</td>
<td>Entropy per unit volume and mass</td>
</tr>
<tr>
<td>$T$</td>
<td>(K)</td>
<td>Temperature</td>
</tr>
<tr>
<td>$u = (u, v)$</td>
<td>(m s$^{-1}$)</td>
<td>Velocity field</td>
</tr>
<tr>
<td>$v_i$, $v_j$</td>
<td>(m s$^{-1}$)</td>
<td>Terminal velocity</td>
</tr>
<tr>
<td>$w_{max}$</td>
<td>(m s$^{-1}$)</td>
<td>Maximum velocity (cf. amplitude of $\psi$)</td>
</tr>
<tr>
<td>$w$</td>
<td>(1)</td>
<td>Averaging weight in particle advection scheme</td>
</tr>
<tr>
<td>$x$, $z$</td>
<td>(m)</td>
<td>Spatial coordinate</td>
</tr>
<tr>
<td>$X$, $Z$</td>
<td>(m)</td>
<td>Domain extent</td>
</tr>
</tbody>
</table>
Code availability

The library is released under 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 is 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++\(^{11}\) and LLVM clang++\(^{12}\) compilers and on Apple OSX with the Apple clang++\(^{13}\) compiler. Consequently, these are considered the supported platforms.

The Supplement related to this article is available online at doi:10.5194/gmd-8-1677-2015-supplement.

Acknowledgements. S. Arabas thanks Shin-ichiro Shima (University of Hyogo, Japan) for introducing the particle-based simulations. We thank D. Jarecka (University of Warsaw) and G. Feingold (NOAA) for insightful discussions and comments on the initial version of the manuscript. We acknowledge contributions to libcloudph++ code from P. Dziekan, D. Jarecka and M. Waruszewski. Development of libcloudph++, libmpdata++ and icicle has been supported by Poland’s National Science Centre (Narodowe Centrum Nauki) (decisions nos. 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 for the work of the developers of the free/libre/open-source software which served as a basis for the implementation of the presented library (see Sect. B for a list). We would like to express our admiration for 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++ continuously benefits from the computational services offered by Travis at their continuous-integration platform.

Edited by: K. Gierens

References


11http://gcc.gnu.org/

12http://llvm.org/

13http://apple.com/xcode

14http://clang.llvm.org/

15http://gnuplot.info/
Khvorostyanov, V. and Curry, J.: Aerosol size spectra and terminal velocities of droplets.


