Cleo: The Fundamental Design of a New Superdroplet Model for High Computational Performance (v0.39.0)

Clara J.A. Bayley[1, 2], Tobias Kölling[1], Ann Kristin Naumann[1, 3, 4], Raphaela Vogel[3], and Bjorn Stevens[1]

[1]Max-Planck-Institut für Meteorologie, Hamburg, Germany
[2]International Max Planck Research School on Earth System Modelling, Hamburg, Germany
[3]Meteorologisches Institut, Universität Hamburg, Hamburg, Germany
[4]Ludwig-Maximilians-Universität München, Munich, Germany

Correspondence: Clara J.A. Bayley (clara.bayley@mpimet.mpg.de)

Abstract. Cleo is a novel implementation of the Super-Droplet Method (SDM) aiming to be efficient on and portable across exascale high-performance computers. It is motivated by the need to develop a SDM that can model warm-cloud microphysics in domains large enough to resolve shallow mesoscale cloud organisation $\mathcal{O}(100\,\mathrm{km})$. This paper introduces Cleo's fundamental structure. First, it explains the choices we made to exploit shared memory spaces on single nodes of high-performance computers. In particular, we discuss our reasoning behind Cleo's ordered and contiguous arrays of structures for superdroplets and grid-boxes, and how we can allocate resources economically and portably through Cleo's MPI domain decomposition, Kokkos library thread parallelism, and coupling to a host dynamical driver. Second, this paper explains how Cleo uses monoids to construct arbitrary combinations and permutations of microphysical processes and data output. These monoids make microphysics and data output highly and easily configurable, which facilitates warm-cloud process understanding. Furthermore, they reduce conditional code branching and enhance Cleo's readability and maintainability. Performance tests on single nodes of Levante HPC System show Cleo has optimal linear scaling with increasing number of superdroplets as well as sufficient memory- and strong-scaling on both CPUs and GPUs to invite CLEO's next stage of development, its optimisation for large eddy simulations on multiple nodes of top-tier and exascale high-performance computers.

1 Introduction

Persistent and large discrepancies between observations of the spatial and temporal patterns of warm-rain and Large Eddy Simulations (LESs) have been ascribed to cloud microphysics for decades (e.g. Randall et al., 2003; Abel and Shipway, 2007; Ackerman et al., 2009; vanZanten et al., 2011) (see also summaries in Khain et al., 2015; Morrison et al., 2020). With the advent of Global Storm Resolving Models (GSRMs) which remove the need to parametrise convection, cloud microphysics is now one of the leading sources of uncertainty in global models too (e.g. Miyakawa et al., 2014; Stevens et al., 2020; Suematsu et al., 2021; Bao and Windmiller, 2021; Lang et al., 2023; Takasuka et al., 2024; Naumann et al., 2025). Much of the uncertainty in both LES and GSRMs arises from epistemic uncertainty, that is uncertainty caused by our insufficient knowledge of microphysical processes and condensate attributes. However, distinct from that, these models also suffer from uncertainty caused by the Eulerian microphysics schemes we conventionally use to represent the knowledge we already have (Morrison

et al., 2020). To tackle uncertainty in both LES and GSRMs we not only need to address fundamental gaps in our knowledge, but we must also reconsider *the way* we represent cloud microphysics in models (Grabowski et al., 2019).

Conventional, Eulerian bulk/moment and bin microphysics schemes have intrinsic deficiencies, for example due to numerical diffusion, the categorisation of condensates into discrete types, and gross assumptions about particle size distributions. The consequent deficiencies in simulated clouds are well documented and contribute to uncertainties in their radiative properties as well as precipitation (e.g. Stevens and Seifert, 2008; Khain et al., 2015; Jian et al., 2021; Schulz and Stevens, 2023). To state just one example, numerical diffusion in physical space is known to activate droplets at cloud-edge and broaden the droplet size distribution, and so may cause unphysical precipitation from stratocumulus (Stevens et al., 1996; Hoffmann, 2016; Morrison et al., 2018; Dziekan et al., 2019). Problems like these are inherent to bulk and bin microphysics schemes and therefore cannot be eradicated by refinement of such models. Indeed, decades of research has aimed to limit the impact of numerical diffusion on cloud microphysics, and yet the problem still persists (Grabowski et al., 2019). If we seek to resolve such issues definitively, then the fundamental limitations of bulk and bin schemes need to be surmounted.

The Super-Droplet Method (SDM; Shima et al., 2009), is a fundamentally different model of cloud microphysics with different limitations and a number of key conceptual advantages (Grabowski et al., 2019). In SDM the cloud condensate population is modelled by "superdroplets". Superdroplets' motion, microphysics and attributes (for example their masses and radii) are similar to what we expect of real condensate particles; however a superdroplet also has a "multiplicity", which expresses how many real particles that superdroplet represents. SDM is a much closer representation of the underlying physics than bulk and bin schemes and, because it is a Lagrangian model, it is non-diffusive (in both physical and mass space) (Morrison et al., 2018; Grabowski et al., 2019). Several SDM implementations already exist, (e.g. Shima et al., 2009; Arabas et al., 2015; Naumann and Seifert, 2015; Brdar and Seifert, 2018; Jaruga and Pawlowska, 2018; Shima et al., 2020; Chandrakar et al., 2021; Bartman et al., 2022; de Jong et al., 2023; Matsushima et al., 2023) and are similar to other particle-in-cell-based cloud microphysics models (e.g. Sölch and Kärcher, 2010; Andrejczuk et al., 2010; Riechelmann et al., 2012; Hoffmann et al., 2015), the key difference being SDM's profound convergence property. As the number of superdroplets increases, the model, including its algorithm for droplet collision-coalescence, tends towards what we expect from explicit individual particle simulations (Shima et al., 2009). Although this does not eliminate the epistemic uncertainty caused by insufficient knowledge of microphysical processes and condensate attributes, SDM's convergence property makes its physical interpretation both conceptually elegant and straightforward. This in turn helps us explore such knowledge gaps.

SDM is also particularly well-suited to trends in high performance computing. Aside from converging, its collision-coalescence algorithm conserves the number of simulated particles, i.e. superdroplets, which is beneficial for load balancing. Moreover SDM is embarrassingly parallelisable because its algorithms for microphysics and motion act independently on individual superdroplets or non-overlapping pairs of superdroplets, which makes it ideal for multi-threaded CPU and GPU architectures and explains impressive scaling with increasing number of superdroplets (Arabas et al., 2015; Bartman and Arabas, 2021; Dziekan and Zmijewski, 2022; Matsushima et al., 2023). Its scaling with increasing microphysical complexity is also better than that of bin models once the number of superdroplet attributes is greater than about four (Shima et al., 2009). This makes SDM better suited to modelling droplets with more complexity, which is rarely done in bin and bulk schemes and is necessary to better

represent important microphysical processes, for example riming in mixed-phase clouds (Brdar and Seifert, 2018) and aerosol chemistry in aerosol-cloud interactions (Jaruga and Pawlowska, 2018). Both the conceptual and computational advantages of SDM make it a promising tool for better modelling of cloud microphysics (Morrison et al., 2020), and in step with the growth of high performance computers (HPCs) we can now use SDM for more ambitious simulations than ever before.

One emerging application for SDM is to model warm-rain in domains large enough to resolve shallow mesoscale organisation $\mathcal{O}(100\,\mathrm{km})$. The evaporation of warm-rain drives cold pool formation and is therefore a key player in mesoscale circulations and cloud organisation (Barnes and Garstang, 1982; Seifert, 2008; Seifert and Heus, 2013; Vogel et al., 2021). Of particular importance in the tropics, mesoscale cloud organisation has major repercussions for the radiative properties of shallow clouds (Nuijens and Siebesma, 2019; Bony et al., 2020) and is highly sensitive to the formation of precipitation (Seifert and Heus, 2013; Yin et al., 2024). There is also ample evidence that mesoscale cloud organisation modulates warm-rain formation (e.g. Stevens et al., 2005; Nuijens et al., 2009; Schulz et al., 2021; Radtke et al., 2022, 2023), and so there are important two-way interactions between the micro- and meso-scale of warm-clouds which, in order to be understood, require an accurate representation of the droplet size distribution, a clear depiction of the microphysical processes at play, and domains large enough to resolve shallow mesoscale circulations. SDM would be an ideal candidate to meet such requirements, except that, presumably limited by computational resources, the largest domains for SDM have not yet exceeded $\mathcal{O}(10\,\mathrm{km})$ (Sato et al., 2017; Chandrakar et al., 2021; Matsushima et al., 2023; Yin et al., 2024) — far smaller than the $\mathcal{O}(100\,\mathrm{km})$ domains needed for resolving mesoscale organisation, and far smaller than that which may be feasible with the growth of exascale high-performance computers, computing at least 10^{18} floating-point operations per second. This is just one example of how, *if* SDM could be made inexpensive enough at large-scale, it would benefit the community who use large-domain explicitly-resolved-dynamics models — including not just LESs, but also GSRMs.

Hence we are creating Cleo: a novel implementation of SDM aiming to model warm-cloud microphysics efficiently on top-tier and exascale high-performance computers. Cleo's name is a tribute to two "super-women", Cleopatra VII Thea Philopator and the pseudonymous mathematician Cleo.

In this first paper we introduce Cleo's fundamental computational structure and explain how it is intended to facilitate warm-cloud process understanding and to exploit single nodes of high-performance computers. Specifically, this paper explains how we exploit the massively parallel heterogeneous resources and the hierarchical memory layout of individual nodes of high-performance computers. Notwithstanding SDM's many advantages in comparison with bulk and bin microphysics, SDM faces the typical challenges of Lagrangian particle models; namely simulations can be memory-intensive and it is time-consuming to transport particles in memory. The particle transport's performance on high-performance computers depends on both its shared- and distributed-memory parallelism, and the distributed-memory parallelism strongly varies not only with Cleo's fundamental structure, but also with the fluid-flow given by the host dynamical driver as well as the computer network topology. In a follow-up paper we will therefore present Cleo's distributed-memory parallelism (MPI domain decomposition) and evaluate its performance specifically for LES across multiple nodes of top-tier and exascale high-performance computers. With regard to Cleo's fundamental computational structure on single nodes, the memory and particle transport expenses are prevalent when accessing, sorting, and shuffling superdroplets. Conscious of this, we chose Cleo's memory layout with the aim of

economically allocating memory and optimising memory access patterns for loops over superdroplets, whilst compromising on other aspects of performance, namely Single Instruction, Multiple Data (SIMD) parallel processing and scalability with increasing number of superdroplet attributes. Secondly, we have sought to maximise Cleo's freedom to allocate resources on massively parallel heterogeneous computer architectures; both through its handling of domain decomposition, which allows SDM and the dynamics to distribute work independently of one-another, as well as through its use of Kokkos for portable thread parallelism (Trott et al., 2022, 2021), in doing so we aim to be able to allocate resources efficiently.

As well as for high performance, Cleo has various features to facilitate warm-cloud process understanding. We took the idea of monoid sets from mathematics and created a computational representation for them which enables adaptive time-stepping, avoids conditional code branching, and makes microphysics and data output highly configurable. The flexibility of microphysics makes sensitivity studies easy to conduct and, for data output, it's easy to target output specifically to ones needs. Also for the purpose of sensitivity studies we made it straightforward to switch Cleo's grid and couple Cleo to different dynamical drivers. The companion to this paper, Bayley et al. (2025), describes the numerical methods for the warm-cloud microphysics and droplet motion which Cleo makes available.

Cleo's fundamental design is described in Sections 2, 3, and 4, with particular emphasis on how it is intended to make SDM simulations suitable for high performance computers (HPCs). In Section 2 we describe Cleo's underlying memory layout and how it is designed for efficient memory management; Section 3 is for how Cleo enables economical resource allocation, and Section 4 describes Cleo's flexibility through monoids. Overviews of Cleo's time-stepping routine and performance on a single node are found in Sections 5 and 6, respectively.

2 Memory Layout

Efficient memory management is principally achieved by Cleo's underlying memory layout of superdroplets and grid-boxes (Sections 2.1 and 2.2). We store superdroplets within each shared memory space in an ordered and contiguous chunk of memory. This optimises access patterns for loops over superdroplets and avoids needing additional buffers which increase memory consumption. However, this layout comes at the cost of sorting superdroplets and increasing cache-misses as the number of superdroplet attributes increases. We explain the details behind this compromise in Section 2.3.

2.1 Grid-Boxes

"Grid-Boxes" (cells) compose Cleo's spatial domain. Each grid-box defines a distinct region of space for the SDM collision algorithm and thereby determines which superdroplets may interact with one another. In some sense the volume of each grid-box therefore determines the accuracy of the SDM because as the volume of each grid-box decreases given a certain number of superdroplets, the resolution of the flow-field for superdroplet motion increases, and the multiplicity of each superdroplet decreases, thus decreasing the errors in the SDM.

Grid-Boxes are structures consisting of three parts as shown in Figure 1a. The "GBx-index" is a unique immutable identifier, typically a constant unsigned integer; the "state" defines the macrophysical properties of the volume, such as the wind

velocity and thermodynamics (temperature, pressure, relative humidity etc.); and the "view" of superdroplets specifies all the superdroplets which occupy the volume at a given time.

The state may change due to microphysical processes, for example condensation lowers the water vapour pressure and increases the temperature, or it may change through information received from the host dynamical driver as explained in Section 3.3.

The view specifies all the superdroplets at a given time whose spatial coordinates fall within the grid-box's boundaries. Rather than actually containing the superdroplets, it is an indicator to their location(s) elsewhere in memory. For example, the view could be a linked-list of pointers to individual superdroplets, or it could be two pointers to the start and end of a sub-section of a larger array of superdroplets (as it is in our current implementation).

Grid-Boxes are agnostic to the underlying grid, meaning they contain no information about their spatial coordinates or their neighbours. This is because LES grids can vary substantially, for example SCALE uses a Cartesian Arakawa-C grid (Sato et al., 2015; Nishizawa et al., 2015), whereas ICON uses a icosahedral-triangular Arakawa-C grid (Hohenegger et al., 2023), and decoupling the microphysics from the grid of the dynamics driver makes it simple to switch between different grids to aid model inter-comparison studies. For a given grid, a collection of maps/functions called "grid-maps" are assembled, to map from a given GBx-index to grid-dependent information. Such maps take a given GBx-index and return for example the coordinates of the boundaries of that grid-box in a particular direction or the GBx-index of the neighbouring grid-box. In doing so they determine the type of grid, the boundary conditions of the model, and whether the model is 0-D, 1-D, 2-D, or 3-D. Thus, changing these properties for different simulations only requires a replacement of the grid-maps. The rest of Cleo, including the grid-boxes themselves, remains unchanged.

2.2 Superdroplets

Each superdroplet in Cleo is a structure composed of four, or optionally five parts, as shown in Figure 1b. The four compulsory parts are its attributes, multiplicity, spatial coordinates, and "SD-GBx-index". The SD-GBx-index always matches the GBx-index of the grid-box whose boundaries enclose the superdroplet's spatial coordinates. It is not a requisite of SDM but assists with keeping the superdroplets ordered by grid-box, which is necessary for efficient cache loading as explained in Section 2.3.

Optionally, each superdroplet can be given an unique identity to facilitate tracking the evolution of individual superdroplets. Alternatively, when tracking is not required, superdroplet identities can be given no unique address. During compilation these identities are optimised out, enabling a simple memory-saving measure especially for simulations involving a very large number of superdroplets. The same optimisation could be applied to grid-boxes' GBx-indexes (since their positions in memory could act as unique identifiers instead) however we keep GBx-indexes for generality.

2.3 Access Patterns

The memory layout of both grid-boxes and superdroplets in a single shared memory space targets efficient memory access patterns. In this layout, the structures for each grid-box are stored together in a single array (an Array of Structures; AoS), and likewise superdroplets are stored in their own AoS, in a contiguous block of memory completely separate from grid-boxes.

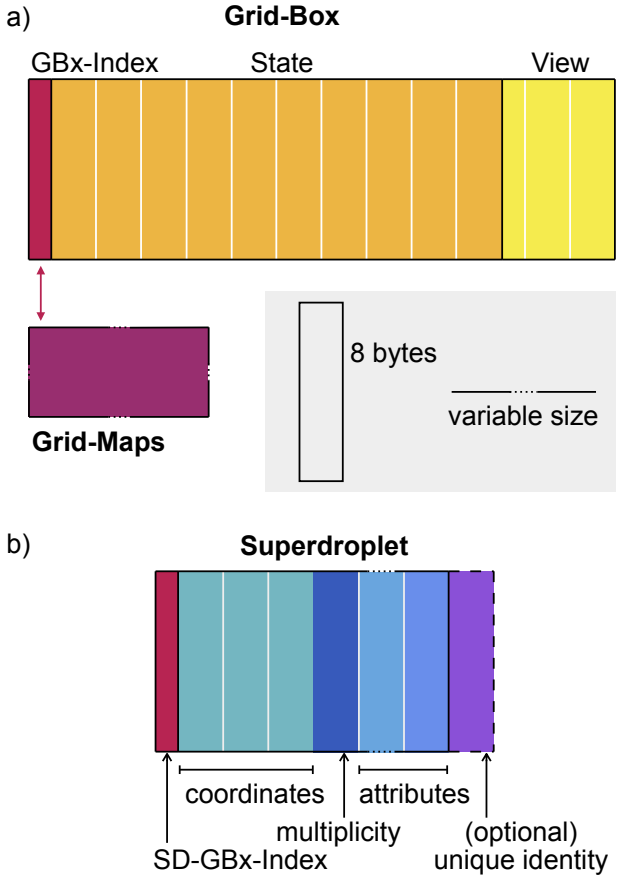

Figure 1. Schematic for the memory layout of a) each grid-box, and b) each superdroplet. The grey box contains a key for the memory consumption of each unit.

Additionally the superdroplets' array is ordered by increasing SD-GBx-index, and hence the view of superdroplets for each grid-box is formed by references to the start and end of the sub-block of superdroplets whose SD-GBx-indexes match the grid-box's GBx-index. This is all summarised in Figure 2. The memory layout we have chosen benefits some computational operations at the expense of others. In particular, ordering the superdroplets by SD-GBx-index makes cache loading efficient but requires the additional expense of sorting them during superdroplet motion, and using AoS favours data transfer over SIMD operations. In the following section we discuss in more detail how these conflicts manifest in the algorithms Cleo uses to conduct SDM microphysics.

Single, contiguous arrays of grid-boxes and superdroplets efficiently handle memory allocations. As opposed to having superdroplets scattered in memory, e.g. by having a separate array of superdroplets for each grid-box, having a single array of superdroplets requires less frequent memory (de)allocations when superdroplets move around the domain, and therefore also avoids the need for buffers which would increase the memory consumption of the model.

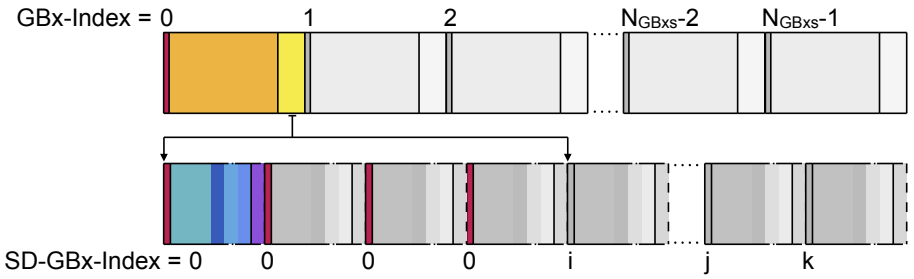

Figure 2. Schematic for the AoS memory layout of grid-boxes and superdroplets. For illustration here the coordinates of the first four superdroplets are within the first grid-box, hence their SD-GBx-indexes match that grid-box's GBx-index and that grid-box's view refers to them.

The layout we have chosen is also computationally efficient for cache loading during loops over grid-boxes and superdroplets — exactly as occurs during microphysics and while updating superdroplets' spatial coordinates — because both grid-boxes and superdroplets are not only contiguous in memory but also ordered. One advantage is that we maximise locality of reference, in contrast to unordered or non-contiguous data. This means that for loops over grid-boxes, cache lines are more fully utilised because the block of data loaded in the caches contains more of the superdroplets required for each grid-box. This applies even during superdroplet collisions, since in SDM only superdroplets in the same grid-box may interact with one another. For loops over superdroplets, also when they are nested within loops over grid-boxes, maximising locality of reference means that the same grid-box is accessed repeatedly in as fast succession as possible. The second major advantage of our ordered and contiguous memory layout is that during parallelised loops over grid-boxes, threads on the same core which act on neighbouring grid-boxes are more likely to access superdroplets in the same data blocks, whereas threads on different cores are more likely to access independent superdroplet data blocks. Both this and data locality result in fewer cache misses and less contention in memory access, which is particularly important for high-performance computers because they are characterised by large cache lines and numerous threads and cores working in parallel. In many respects our memory layout amounts to cache blocking, which can help to exploit the embarrassingly parallel nature of SDM, as was shown by Matsushima et al. (2023), who employed cache blocking, albeit differently, to drastically improve the performance of another SDM implementation. As such, Cleo is designed for both efficient memory allocation and access patterns for SDM on high-performance computers.

Whilst keeping the superdroplet array ordered by SD-GBx-index makes loops over superdroplets highly efficient, it also means that during superdroplet motion we must sort the array, as illustrated in Figure 3. Superdroplets' spatial coordinates and hence SD-GBx-indexes may change, breaking the crucial condition for cache blocking that superdroplets occupying the same grid-box exist in a contiguous chunk of memory. We therefore apply a counting-sort algorithm to the superdroplet array as the last step of superdroplet motion. Although this algorithm scales linearly with increasing number of superdroplets, it nevertheless involves time-consuming data copies, scatter-gather patterns and/or atomic operations, and it doubles the memory consumption of the superdroplet array. Choosing to keep the superdroplet array ordered is therefore a trade-off. On the one

hand it makes loops over superdroplets more efficient, on the other hand it makes superdroplet motion more expensive due to sorting.

Using an AoS for superdroplets rather than a Structure of Arrays (SoA) prioritises efficient data movement over vectorisation. With the AoS layout, irrespective of the number of attributes of each superdroplet, the speed of algorithms which sort/shuffle superdroplets during motion/microphysics scale only with the number of superdroplets. In contrast any SoA layout which still maintains the advantages for memory access patterns of ordered superdroplet data would need to perform sorting/shuffling on the array for each individual sub-component of the superdroplets separately, resulting in more computations. Another strength of AoS over SoA is that the addition or removal of superdroplets during motion involves less memory (de)allocations and more efficient cache loading. However, with the AoS layout, operations that act on a single attribute can be less efficient because SIMD parallel processing is harder to apply. Also if the number of attributes of each superdroplet is increased, the number of cache misses likewise increases.

Which data layout results in better performance certainly depends on the computer architecture (e.g. cache size and types of processors) as well as the number of superdroplets and number of superdroplet attributes as a result of the compromise between fast computations through vectorisation and fast memory access patterns. For simulations involving many superdroplets (lots of data transfer) with few attributes (few cache misses), the memory layout we have chosen should be more suitable, but rigorous benchmarking would be needed to confirm the preferred layout for a given setup. In the end, some compromise between the two (an Array of Structures of Arrays, AoSoA) may be the better than either extreme. The decision for the memory layout of Cleo is nevertheless bolstered by the work of Matsushima et al. (2023), who also focused on data movement to optimise their SDM's performance, as well as by the rising weight given to the High Performance Conjugate Gradients Benchmark when ranking HPCs, which demonstrates the increasing concern for the performance of memory bandwidth limited computations.

3 Resource Allocation

Cleo has the potential to allocate resources economically because of its parallelism and the way it couples to a host dynamical driver. We used Kokkos to implement thread parallelism and Message Passing Interface (MPI) for domain decomposition. As such, Cleo can make use of the available memory hierarchies and execution resources on a diverse set of HPC architectures. Whereas Cleo advects superdroplets according to the fluid flow (wind fields) itself, to advect thermodynamic fields (temperature, pressure etc.) Cleo must be coupled to a host dynamical driver capable of advection. However the domain decomposition and computational resources for the dynamics can be independent from those used by SDM. This allows us greater freedom to improve load balancing and therefore to run economical simulations.

3.1 Kokkos Thread Parallelism

Cleo depends on Kokkos for performance portable thread parallelism across a diverse range of computer architectures. Kokkos is a collection of libraries which enable us to write a single source code that builds for an arbitrary number of CPU cores and optionally GPU cores (Edwards et al., 2014). During compilation on a given computer architecture, Kokkos evaluates

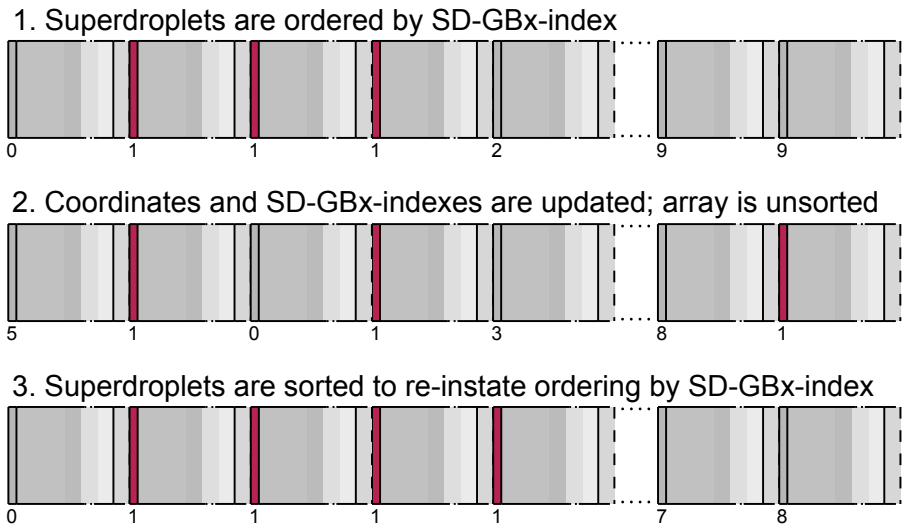

Figure 3. Schematic of the superdroplet array before, during and after superdroplet motion. To illustrate the ordering, the superdroplets have been given SD-GBx-indexes for grid-boxes from 0 to 9.

the hardware available and optional user-defined specifications, to determine how to appropriately allocate the memory and execution resources to achieve high performance. For example, it determines whether to use column- or row-major layouts for arrays of grid-boxes and superdroplets in order to avoid cache misses, and it chooses task sizes for threads of parallelised loops which optimise cache blocking. By doing so, Kokkos implements (hierarchical) parallelism of loops over grid-boxes and superdroplets and selects memory layouts designed to be efficient for a specific processor or accelerator. (Alternatively the memory layouts and task distribution can be manually specified.) When loops over superdroplets are nested inside loops over grid-boxes, as occurs during microphysics and superdroplet motion but not necessarily during initialisation or data output, hierarchical parallelism is invoked; that is both the outer loop over grid boxes and the inner loop over superdroplets are parallelised. The exception is if Kokkos (or manually the user) determines that for the given hardware available the number of grid-boxes or superdroplets is too small for it to be worthwhile to parallelise the loop.

The key benefits of using Kokkos are that in abstracting the parallelism, we are able to separate the software's purpose from the details of a computer's hardware. Thus from one source code we can provide specialised thread parallelism at the same time as portability and maintainability across many architectures. At the time of writing, we have tested CUDA, OpenMP and C++Threads, and the full list of available parallel execution spaces also includes OpenMPTarget, HIP, SYCL and HPX.

3.2 MPI Domain Decomposition

For distributed memory parallelism Cleo uses MPI domain decomposition. The layout described in Section 2 is replicated for the sub-domain given to each MPI process and an extra step in superdroplet motion between the second and third steps of Figure 3 is activated to send superdroplets between processes. This is the only MPI communication Cleo requires for its domain

decomposition during time-stepping. To choose the decomposition, Cleo first calculates all the possible ways grid-boxes could be distributed across MPI processes such the number of grid-boxes on each process is as equal as possible, then Cleo chooses the domain decomposition from these options which keeps the most vertical columns on the same MPI process. Since the most frequent motion of superdroplets is usually vertically, by choosing this domain decomposition we typically avoid excessive communication compared to the other options which also evenly distribute the number of grid-boxes in the domain.

This is the first implementation of MPI parallelism in Cleo and is certainly not the most optimal. Since the workload on each MPI process actually scales to first order with the number of superdroplets, weighting the domain decomposition by the number of superdroplets in each grid-box, rather than the number of grid-boxes, would be the first improvement one could make to achieve better load balancing. Further refinement could also determine the domain decomposition by estimating the activity of the superdroplets too, based on the idea that superdroplets predicted to be in or near cloudy regions of the domain would generally be expected to collide more frequently and grow/shrink more rapidly via condensation/evaporation, and hence require more floating-point operations than superdroplets in non-cloudy regions. The major challenge with such superdroplet-based domain decompositions is that superdroplets (and cloudy regions) move around the domain and superdroplets may be added or removed depending on a simulation's boundary conditions (e.g. its aerosol/superdroplet sources and lateral superdroplet sinks). The consequent changes in the number and activity of superdroplets changes the workload of MPI processes and a better optimised domain decomposition would dynamically adapt to restore balance. One well-established way of dynamical load-balancing in Lagrangian particle-based methods is using Hilbert space-filling curves to calculate the most evenly balanced partitioning of the total workload across MPI processes. In Cleo, any superdroplet-based domain decomposition would need to obey the added constraint that superdroplets in the same grid-box of the domain remain contiguously ordered in memory, in order to benefit the memory access patterns on each node, as discussed in Section 2.3.

The performance of any optimisation of Cleo's MPI domain decomposition strongly depends on the fluid flow given by the host dynamical driver as well as the computer network topology. The fluid flow determines the degree to which superdroplet movement unbalances the load on each MPI process for a given domain decomposition. The computer network topology determines the speed (cost) of MPI communication, which influences the optimal size of the workload given to each process and the acceptable degree of load imbalance for efficient resource allocation. We therefore reserve optimisation of Cleo's MPI domain decomposition and its performance evaluation for a follow-up paper which discusses them in the context of realistic flows and coupling to a specific dynamical driver on top-tier and exascale high-performance computers.

3.3 Coupling Cleo to a Host Dynamical Driver

Cleo is designed to run concurrently to a host dynamical driver called a "dynamics-solver" and exchange information with it via a "dynamics-coupler". In a one-way coupling, Cleo receives information for the state of each grid-box from the dynamics-solver, whereas in a two-way coupling, Cleo also sends state information back. For illustration, a simple one-way coupling is when thermodynamics at a particular time-step are received from a dynamics-solver which stores arrays read from a file. Given the information stored in the state of each grid-box (wind velocity and thermodynamics), Cleo can enact microphysical processes on the superdroplets in each grid-box and it can move superdroplets according to Figure 3: first updating their

coordinates and SD-GBX-Indexes and then re-ordering the superdroplet arrays. The numerical methods for microphysics and for updating superdroplet coordinates are provided in Bayley et al. (2025). Whilst Cleo can thus advect superdroplets, to advect thermodynamic fields as per the fluid flow, Cleo must be coupled to a dynamics-solver with a fluid-dynamical core.

The dynamics-coupler determines the degree of independence of the grid and of the resource allocation for Cleo and the dynamics-solver. For the maximum possible flexibility, Cleo can be coupled to a dynamics-solver through Yet Another Coupler (YAC; Hanke et al., 2016). YAC uses MPI communication and can also interpolate variables between different grids meaning that, by using YAC, Cleo and the dynamics-solver can not only have different domain decompositions, but also have different grids. Using different grids is advantageous because it enables the dynamics-solver to compose the domain in the optimal way for its fluid-dynamics. Meanwhile Cleo can compose the domain more favourably for SDM, for example using a nested grid, or grid boundaries which reduce grid-box volumes and/or simplify the numerics of superdroplet motion. In general, using MPI means Cleo and the dynamics-solver do not have shared memory and that their domain decompositions are independent. Whilst this can result in costlier communication, it also maximises our freedom to optimise the load balancing and so economise the allocation of computer resources.

4 Flexibility through Monoids

Cleo uses monoids to enable model flexibility without added run-time from conditional code branching. In abstract mathematics, a monoid is a closed set with an associative binary operation and an identity element. Put simply, monoid sets are composed of elements (members) which result in another element of the set when they're added pairwise together, and the identity element is defined such that the outcome of its addition with another element is just that other element unchanged. Acrylic paint is one example of a monoid set, whereby different colours are the elements of the set and transparent acrylic paint is the identity element. Mixing two acrylic paints together is their associative binary operation (in this special case, also commutative).

We apply the idea of monoids computationally by creating objects in code that can be combined pairwise and associatively with one-another. To do this, first we define a set of constraints a code-object must satisfy in order to be allowed to be combined with another such object. In other words, we define constraints an object must satisfy to be part of a certain code-based monoid set. Secondly, alongside these constraints, we define a function (an associative binary operation) which determines exactly how two objects which obey these constraints are combined together such that the return of the function is a new object which itself satisfies the given constraints. In this way, any number and permutation of such objects can be combined sequentially to create the final object that is used at run-time.

As an example, consider some code which calls a function "do_microphysics" through an object called "Z". Z can be any object as long as it obeys some constraints: it contains a function called "do_microphysics", which takes superdroplets as an argument and potentially modifies them. Now let us create any number of objects, A, B, C, ... which all obey these constraints but have different implementations of the "do_microphysics" function. For example A.do_microphysics may model condensation, whereas B.do_microphysics may model collision-coalescence. We then define an associative binary operation for these

objects that returns a new one which likewise satisfies the constraints of the "do_microphysics" function. For example the operation could return "AB" from A and B where AB.do_microphysics = A.do_microphysics followed by B.do_microphysics. Note that since monoids are not necessarily commutative, "AB" is allowed to cause different microphysical outcomes to those of "BA" — in this example, the growth of superdroplets by condensation before collisions can result in different outcomes to collisions followed by condensation (i.e $AB \neq BA$). Since we chose to define the outcome of the associative binary operation (AB) to also satisfy the constraints of the "do_microphysics" function, we can then combine AB with further objects, C, D, etc. to create one final object, Z, whose "do_microphysics" function is an assembly of all the microphysics we desire (again, associatively but not necessarily commutativity). Everywhere else in the code we use Z, for example to call the "do_microphysics" function (rather than A, B, or C etc.), meaning that if we wish to change the microphysics we need only recompile and run the model with a different combination of things comprising Z. This allows us extraordinary freedom regarding the implementation of the "do_microphysics" function whilst keeping the rest of the code intact.

To implement monoids in Cleo, we take advantage of template meta-programming with C++20 concepts. Specifically, for a given monoid set we use a C++20 concept to impose the constraints of the set on templated types, and we overload the right-shift operator (`operator>>`) to implement the function for the associative binary operation. We then create structures/-classes which obey the C++ concept and can thus be combined during compilation, in whatever way desired, to instantiate the templated type that is used throughout the code at run-time. Using C++20 concepts is not essential, but has the added benefit of more comprehensible error messages than ordinary template meta-programming in C++.

There are many other plausible ways to express monoids in code. Within older C++ standards (pre-2020) one could instead use for example a base class, a trait, or even just constraints written on a piece of paper. However, base classes would create more restricted, closed definitions of monoids, and the latter options would be more verbose and error-prone. Based on how we defined computational monoids above, other compiled or just-in-time compiled languages could also implement monoids computationally with the same overall behaviour as our monoids in C++. Naturally, languages with object-oriented programming paradigms are better-suited to implementing monoids since monoids are fundamentally described by set theory.

Using monoids to enable model flexibility provides faster run-time performance than using conditional code branching. The decisions regarding which functions are called occurs upon instantiation of each templated type during compilation. This allows compilers to efficiently inline and optimise the used code, whilst at the same time increasing code locality by not compiling unused code. More importantly, the monoidal code is in stark contrast to a code which makes decisions at run-time, for example by using if-statements to determine whether or not to call functions. Indeed, demanding the same degree of flexibility from a code which uses if-statements would require an excessive number of conditional branches — in the example above, to be able to model just A, B, AB, or BA requires three. If this were implemented in Cleo, each branching would add computational expense and reduce the code's modularity. Modular as opposed to monolithic code is more readable and easier to maintain because it can be understood in piecewise units and each unit can be tested and developed independently, as well as concurrently. Although at the time of writing Cleo does not yet have unit tests for its monoids, they are still modular units of code and thus facilitate code readability and maintainability.

Listing 1 The definition of a microphysical process in Cleo

```
/* define a microphysical process */
template <typename P>
concept MicrophysicalProcess =
    requires(P p, TeamMember &tm, unsigned int t, view_supers supers,
             State &state, NullMonitor mo) {
      { p.next_step(t) } -> std::convertible_to<unsigned int>;
      { p.on_step(t) } -> std::same_as<bool>;
      { p.run_step(tm, t, supers, state, mo) } -> std::same_as<void>;
    };
```

Thus, to ensure run-time performance whilst still enabling maintainability and flexibility, Cleo defines monoids for microphysics and data output. Section 4.1 details our monoid for microphysical processes, namely the C++ concept which defines microphysical processes, the structure which acts as the identity element of the set, and the right-shift operator used to combine microphysical processes with one another. Likewise, Section 4.2 details the monoid set for "observers", the elements of which can be used for data output.

4.1 Microphysical Processes

Microphysics in Cleo is enacted by the instantiation of a templated type which satisfies the constraints of a "microphysical process". The constraints are chosen to ensure a type instantiated to perform microphysics will function in the time-stepping routine displayed in Figure 4a. They are that every microphysical process has three (GPU compatible) functions, `next_step`, `on_step`, and `run_step`, each with specific signatures as imposed by the C++20 concept shown in Listing 1. Naturally, types which count as valid microphysical processes may contain additional information beyond these three functions, e.g. a time-step value or a random number generator for a specific microphysics algorithm, however such additional information is not a requirement of a microphysical process.

We also choose to define the associative binary operation $A \oplus B = C$ for types that obey the constraints of a microphysical process such that:

- `C.next_step = minimum(A.next_step, B.next_step),`
- `C.on_step = (A.on_step or B.on_step),`
- `C.run_step = A.run_step, then B.run_step.`

This definition enables us to compile a single microphysical process that obeys the time-stepping routine in Figure 4a but is actually constructed from the sum of any permutation of microphysical processes, as demonstrated in Listing 2. The flow diagram between C.run_step and C.next_step in Figure 4a is simply supplanted by calling the run_step function of each of the original microphysical processes sequentially, as exampled by Figure 4b.

Listing 2 Example code showing how microphysical processes are combined in Cleo. Here we choose "$\gg$" to be the operator in the definition of $A \oplus B = C$ for microphysical processes.

```
/* define how to combine microphysical processes */
template <MicrophysicalProcess M1, MicrophysicalProcess M2>
struct CombinedMicrophysicalProcess {
  M1 a; /**< First microphysical process. */
  M2 b; /**< Second microphysical process. */

  unsigned int next_step(const unsigned int t) const {
    return Kokkos::min(a.next_step(t), b.next_step(t));
  }

  bool on_step(const unsigned int t) const {
    return a.on_step(t) || b.on_step(t);
  }

  void run_step(const TeamMember &tm, const unsigned int t, view_supers supers,
                State &state, const Monitor auto mo) const {
    a.run_step(tm, t, supers, state, mo);
    b.run_step(tm, t, supers, state, mo);
  }
};

auto operator>>(const MicrophysicalProcess auto a,
                const MicrophysicalProcess auto b) {
  return CombinedMicrophysicalProcess{a, b};
}

// ... ///

/* use a combination of microphysical processes */
const MicrophysicalProcess auto mphys1 = condensation(config);
const MicrophysicalProcess auto mphys2 = collision_coalescence(config);
const MicrophysicalProcess auto mphys3 = collision_breakup(config);
const MicrophysicalProcess auto microphysics =
    mphys3 >> mphys2 >> mphys1;  // change this line to change microphysics
}
```

By defining the addition of microphysical processes in this way, we also ensure adaptive time-stepping in which the combined microphysical process always takes the largest possible time-step such that the time-steps of original processes are respected. Consider the microphysical process $C = A \oplus B$. The time-stepping routine in Figure 4a starts at time=0 s, which is less than the final time for time-stepping, t_final. When C.run_step is first called, both A.on_step and B.on_step return false and afterwards C.next_step updates the time to the smaller value out of A.next_step and B.next_step. On the next loop iteration, the microphysical process(es) out of A and B which return true from their own on_step function enact microphysics, C.run_step then returns and time is incremented again. The loop for microphysics enacts this logic repeatedly until t_final is

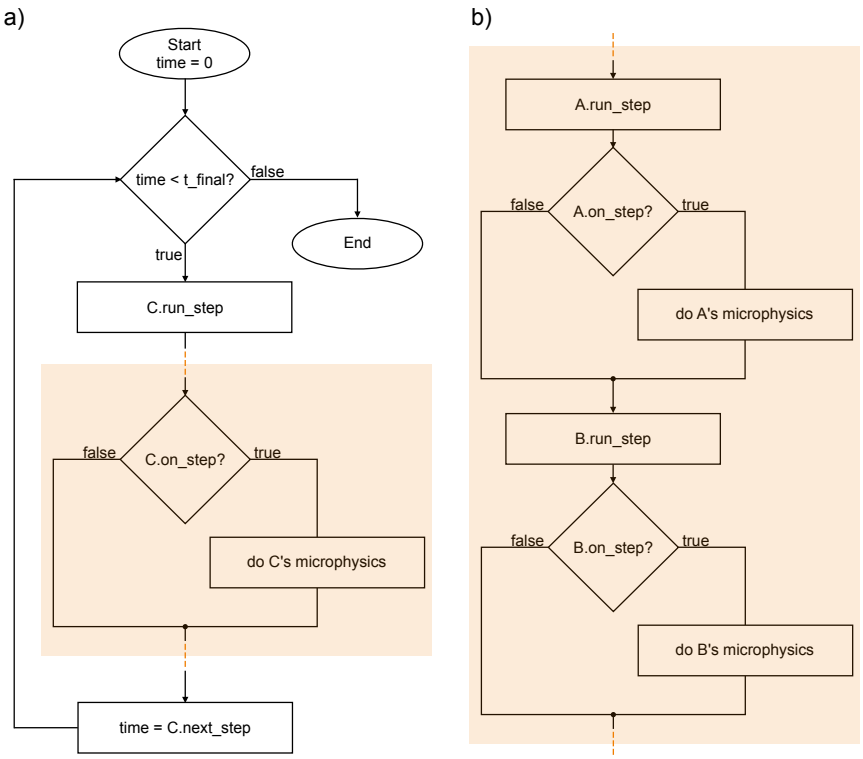

Figure 4. a) The flow diagram for the time-stepping routine of some microphysical process "C" as described by the example in the text. b) The flow diagram between C.run_step and C.next_step when $C = A \oplus B$.

exceeded and the simulation terminates. For demonstration, if t_final=20s and A and B have constant time-steps of $3\,\mathrm{s}$ and $5\,\mathrm{s}$, respectively, time advances with the letter in the bracket indicating which microphysics would be enacted at each time-step in the following sequence: $0\,\mathrm{s}$ -> $3\,\mathrm{s}$ (A) -> $5\,\mathrm{s}$ (B) -> $6\,\mathrm{s}$ (A) -> $9\,\mathrm{s}$ (A) -> $10\,\mathrm{s}$ (B) -> $12\,\mathrm{s}$ (A) -> $15\,\mathrm{s}$ (A and B) -> $18\,\mathrm{s}$ (A) -> $20\,\mathrm{s}$ (B). In general, A and B need not have constant time-steps and can themselves be formed from the summation of other microphysical processes. Thus we can construct a single microphysical process during compilation which is the sum of any permutation of microphysical processes and enables adaptive time-stepping which adheres to each process' time-steps.

For completeness of the monoid for microphysical processes, the set has an identity element, `NullMicrophysicalProcess` shown in Listing 3. This is useful as an initial microphysics construction, or in simulations when no microphysics is desired.

4.2 Observers

Data output, for example writing information to non-volatile memory or statements to a terminal window, is entirely analogous to microphysics. Output is done by the instantiation of a templated type which satisfies the constraints of an "observer" - a concept for things which have the possibility to read and copy information from Cleo at selected points during time-stepping.

Listing 3 The identity element of the microphysical process monoid in Cleo.

```
/* define the identity element for a microphysical process */
struct NullMicrophysicalProcess {
  unsigned int next_step(const unsigned int t) const {
    return std::numeric_limits<unsigned int>::max();
  }

  bool on_step(const unsigned int t) const { return false; }

  void run_step(const TeamMember &tm, const unsigned int t, view_supers supers,
                State &state, const Monitor auto mo) const {}
};
```

Listing 4 The definition of an observer in Cleo

```
/* define an Observer */
template <typename Obs>
concept Observer = requires(Obs obs, unsigned int t, view_constgbx gbxs,
                            view_constsupers supers) {
  { obs.before_timestepping(gbxs) } -> std::same_as<void>;
  { obs.after_timestepping() } -> std::same_as<void>;
  { obs.next_step(t) } -> std::convertible_to<unsigned int>;
  { obs.on_step(t) } -> std::same_as<bool>;
  { obs.at_start_step(t, gbxs, supers) } -> std::same_as<void>;
  { obs.get_monitor() };  // should return a Monitor type
};
```

Structures called "Monitors" are used by some observers for output which cannot be obtained at the start or end of a time-step, but rather must be monitored during a SDM time-step — for example the mass change due to condensation/evaporation. Monitors are also a monoid in Cleo, but we omit the set's definition here for the sake of brevity. The constraints on an observer are chosen to ensure the type instantiated as the observer will function in Cleo's time-stepping routine shown in Figure 5 (pink boxes). They are that every observer has six functions, `before_timestepping`, `after_timestepping`, `next_step`, `on_step`, `at_start_step`, and `get_monitor`, each with specific signatures as imposed by the C++20 concept shown in Listing 4.

Any type which satisfies these constraints is a valid element of the monoid for observers and can therefore be combined with another element using the associative binary operation $A \oplus B = C$, which we choose to define as:

- `C.before_timestepping = A.before_timestepping, then B.before_timestepping,`
- `C.after_timestepping = A.after_timestepping, then B.after_timestepping,`
- `C.next_step = minimum(A.next_step, B.next_step),`

- `C.on_step = (A.on_step or B.on_step)`,
- `C.at_start_step = A.at_start_step, then B.at_start_step`,
- `C.get_monitor = A.get_monitor` $\oplus$ `B.get_monitor`,

as demonstrated in Listing 5.

Exactly like, but independent of, microphysical processes, the sum of different observers ensures adaptive time-stepping. Likewise this means observers can have different output frequencies and ones that are not necessarily factors of each other. In fact, the constraints on the time-stepping functions for observers and microphysics are so similar they could stem from the same parent time-stepping concept, but for simplicity we keep them distinct.

Observers that write data to non-volatile memory are designed to write large datasets efficiently. Buffers hold output data in RAM until they reach a specified chunk size ($\sim$10 MB by default). These are then written to chunks of arrays in a dataset according to the Zarr storage specification version 2. Zarr is chosen because it is well-suited to high throughput of large N-dimensional arrays, especially because it enables parallelised reading/writing. The dataset is compliant with the requirements of Xarray, and is therefore also compatible with NetCDF. Data can be written as any data-type of the NetCDF Climate and Forecast (CF) Metadata Convention, but to save memory it is cast to 4-byte types by default. Additionally, arrays of superdroplet data (e.g. their spatial coordinates) are output according to the CF Metadata Convention's contiguous ragged representation. This is because the number of superdroplets may change over time if superdroplets leave or enter the domain and so the length of data at each output time-step may change. A ragged array, as opposed to a fixed shape array with dimensions of the largest length of data, does not contain empty data points and so saves memory.

For completeness, the monoid for observers has an identity element, `NullObserver`, shown in Listing 6. This is useful as an initial observer construction, or in simulations which do not require output from Cleo.

5 Time-stepping Cleo

Cleo's entire time-stepping routine is shown in Figure 5. In summary, one time-step consists of seven key stages:

1. Determine the next time-step, t_next.
2. If on a coupling time-step, receive thermodynamics and wind velocity fields for SDM from the dynamics-solver via the dynamics-coupler.
3. If on an observation time-step, perform observation.
4. Time-step the dynamics-solver from the current time to t_next.
5. Time-step SDM from the current time to t_next via the sub-time-stepping routine shown in Figure 5b, enacting microphysics followed by superdroplet motion.

6. If on a coupling time-step, send thermodynamics from SDM to the dynamics-solver via the dynamics-coupler.

7. Increment the time to t_next.

The length of one time-step and the exact implementation of each of these parts is determined by types instantiated for the dynamics-solver, dynamics-coupler, grid-maps, observer, superdroplet motion, and microphysical process. For example, when using a one-way coupling the dynamics-coupler's function for step 6 is empty. The sub-time-stepping routine for SDM shown in Figure 5b is simply an extension of the logic from Figure 4a to include observers and superdroplet motion as well as microphysics. Although currently it is performed afterwards, to reduce run-time the entire sub-time-stepping routine could in principle be made asynchronous to the time-step of the dynamics-solver.

6 Performance

Here we show the performance of Cleo's superdroplet-scaling and strong-scaling using OpenMP, C++Threads and CUDA parallelism. The simulations are performed on a single node of Levante HPC at the German Climate Computing Center (DKRZ) using 1, 16, 64 or 128 CPU threads and, when CUDA is enabled, an additional GPU[1]. The node's configuration and the software used are listed in Table 1. To change the problem size we decrease the volume of each grid-box, holding the domain volume and the number of superdroplets per grid-box fixed (Matsushima et al., 2023) (as opposed to fixing the grid-box volumes and increasing the number of superdroplets per grid-box; Dziekan and Zmijewski, 2022). This not only decreases the variance in the number of droplet collisions and increases the precision of the droplet size distribution, but also increases the resolution of the flow field and the accuracy of each superdroplet's position (Bayley et al., 2025). The better this superdroplet-scaling, i.e. grid-box scaling, the more efficiently we can run accurate simulations.

The simulations are of a 3-D Cartesian domain $6000\,\mathrm{m} \times 300\,\mathrm{m} \times 1500\,\mathrm{m}$ with 256 superdroplets per grid-box, varying grid-spacing as listed in Table 2 and with the initial conditions shown in Figure 6a-d. They include condensation, evaporation, collision-coalescence and droplet motion using the configuration listed in Table 3 for the physics described in Bayley et al. (2025). The droplets in every grid-box are initialised from a bimodal log-normal dry-aerosol distribution of NaCl. The modes are at $20\,\mathrm{nm}$ and $150\,\mathrm{nm}$ with standard deviations of 1.4 and 1.6, respectively, their relative probability is 3:2, and the total droplet number concentration is $500\,\mathrm{cm}^{-3}$. Superdroplets are created by randomly sampling 256 evenly-in-log-space bins between $5\,\mathrm{nm}$ and $1\,\mu\mathrm{m}$ — as in the "Single-SIP-init" method of Unterstrasser et al. (2017), but in radius not mass space. Superdroplets' spatial coordinates are uniformly randomly sampled from the space within each grid-box. The dynamics are a simplified version of the 2-D kinematic flow model from Arabas et al. (2015). To extend the simulation into 3-D, we replicate the 2-D (x-z) flow uniformly along the additional (y) dimension. We prescribe the same divergence-free wind field and hydrostatic equilibrium, but the kinetics are held constant in time, without feedback from microphysics nor relaxation towards the initial profiles. To mimic sub-cloud and in-cloud conditions, the temperature has a lapse rate of $9.8\,\mathrm{K\,km}^{-1}$ below $750\,\mathrm{m}$ and $6.5\,\mathrm{K\,km}^{-1}$ above, while the water vapour has a lapse rate of $2.97\,\mathrm{g\,kg}^{-1}\,\mathrm{km}^{-1}$ below $750\,\mathrm{m}$ and super-saturation is fixed

[1]Tests with CUDA parallelism also used OpenMP for parallelised CPU code, e.g. some parts of initialisation. Since the majority of the time is spent on GPUs, changing the number of CPU threads has negligible effect on the CUDA build's speed-up and so we only show plots for the tests with 128 CPU threads.

at 0.1% above. The surface pressure is $1013.15\,\mathrm{hPa}$, the surface temperature is $297.9\,\mathrm{K}$, and the surface water vapour mass mixing ratio is $16\,\mathrm{g\,kg^{-1}}$. We ran the simulations for $80\,\mathrm{mins}$ without data output and present results per $1\,\mathrm{s}$ of simulated time. An example of the superdroplets' evolution up-to the end of the simulation is shown in Figure 6e.

The superdroplet-scaling of the wall-clock time is shown in Figure 7 for the maximal set of resources for each build: serial (1 thread), CUDA (128 CPU threads and 1 GPU), OpenMP (128 threads) and C++Threads (128 threads). The wall-clock time for all the builds is almost entirely time-stepping SDM, as opposed to initialisation or the dynamics-solver, and time-stepping SDM scales linearly with increasing number of superdroplets once the total number of superdroplets is large enough. This is because the bottleneck of microphysics is the random shuffling of superdroplets required by the method for SDM collisions, and the bottleneck of motion is the aforementioned counting-sort algorithm (Section 2.3). Both of these algorithms scale linearly with increasing number of superdroplets. For C++Threads and OpenMP builds, motion (i.e. sorting) is the dominant bottleneck, while for serial and CUDA builds, it is the microphysics (i.e. shuffling). The time-per-call of motion is, however, three times greater than shown here since the time-step was $3\,\mathrm{s}$. The expense of motion per-call is therefore comparable to that of microphysics for all builds, and considerably more for OpenMP and C++Threads.

The strong-scaling in Figure 8 shows the speed-up of the wall-clock time with increasing number of CPU threads for various problem sizes[1]. Overall, none of the builds reach optimal efficiency (Figure 8a); CUDA speed-up is limited by microphysics (Figure 8b), while OpenMP and C++Threads speed-ups are limited by motion (Figure 8c).

The limit for CUDA parallelism due to microphysics can be seen in Table 4. The speed-up of motion maximizes at 150 for the largest problem sizes, however microphysics appears to plateau at smaller problem sizes and at 70. The bottleneck is again the shuffling algorithm we use during collision-coalescence, presumably due to the relatively slow clock-speed of a single GPU thread during the serial step of the Fisher-Yates algorithm we use for shuffling. A different (possibly parallelised) algorithm may therefore improve performance, or alternatively, the shuffling could be performed concurrently with other parts of the time-stepping routine. For OpenMP, the strong-scaling of microphysics is almost perfect, at 100% efficiency for large enough problem sizes, however, the overall speed-up is limited by the speed-up of motion which is no greater than 8. The scaling of C++Threads is better than OpenMP despite slightly less-optimal scaling of the microphysics because motion scales better for all but the largest problem size. The bottleneck of motion on CPUs is not the data copying itself, but rather the synchronisation of threads required by scatter-gather patterns in the counting-sort algorithm. Data copying nevertheless doubles the memory footprint of the superdroplets. A more specialised sorting algorithm which exchanges rather than copies superdroplets and uses the relationship between the GBx-indexes to avoid scatter-gather patterns would therefore provide better performance both in terms of memory usage and speed. Our results are in agreement with Matsushima et al. (2023) who found that after optimising their SDM, superdroplet tracking (movement and sorting) was indeed the bottleneck of SDM on CPUs, and for the largest simulation this was predominately due to superdroplet sorting.

The bottlenecks in Cleo's microphysics and motion are expected to become less acute with more complex simulations. The addition of more microphysical processes, for example to model frozen condensates, will increase the cost of microphysics in ways that decrease importance of the shuffling algorithm. Coupling Cleo to a dynamics-solver capable of advection will diminish SDM's role in the total run-time and introduce other dynamics-related bottlenecks. Above all, the expense of transporting

superdroplets between MPI processes is expected to make the cost of sorting during superdroplet motion inferior to the cost of MPI domain decomposition, even once optimised.

Figure 9 shows the superdroplet- and strong-scaling of Cleo's memory consumption is as expected. The memory footprints increase linearly with increasing number of superdroplets and are almost constant with increasing CPU threads. The overhead from using OpenMP or C++Threads parallelism is almost insignificant. It is larger for smaller problem sizes but still no greater than 1%, and is predominately from the creation of thread-safe parallel random number generators and arrays for the scatter-gather patterns. Due to the large number of threads on a GPU, the overhead of CUDA parallelism for problem sizes less than approximately 10^6 superdroplets can be relatively large because thread-safe parallel random number generators are more expensive than the superdroplets and grid-boxes themselves. For large enough problem sizes however, the overhead of CUDA is also insignificant, and the maximum memory allocation (i.e. the superdroplet memory allocation) can actually be substantially smaller than in serial. The high-water memory consumption, that is the largest resident set size memory used, can be divided into the apparent memory consumed by a single superdroplet and grid-box. As demonstrated in Table 5, for all builds both superdroplets and grid-boxes are effectively $\mathcal{O}(100\,\mathrm{bytes})$, whereas by default stand-alone they are $68\,\mathrm{bytes}$ and $120\,\mathrm{bytes}$, respectively.

7 Conclusions

To satisfy our curiosity about cloud responses to climate change, SDM offers a promising new way to model cloud microphysics. SDM simulations, even in the absence of collision-coalescence, overcome many of the problematic features of conventional microphysics representations and this has already proven fruitful, for example in studying the role of turbulence during droplet growth (Li et al., 2018, 2020; Grabowski and Thomas, 2021; Chandrakar et al., 2024; La et al., 2025). However, the computational cost currently inhibits SDM from certain areas of research, an important case being the study of interactions between cloud microphysics and their mesoscale organisation. This research requires LES that can clearly depict warm-cloud microphysical processes and resolve droplet sizes whilst also capturing mesoscale circulations. We therefore need a SDM for studying warm-clouds in LES with hectometre resolution and $\mathcal{O}(100\,\mathrm{km})$ domains. Cleo is a new computational implementation of SDM intending to be fit for that purpose. Of course, to make such large-domain LES feasible, Cleo must be able to harness both the shared- and distributed-memory parallelism of exascale computers, and whilst the shared-memory performance depends mostly on the fundamental structure of the code, the distributed-memory parallelism is also sensitive to the fluid-flow given by the host dynamical driver and the computer network topology. As such, we will evaluate Cleo's distributed-memory parallelism in an follow-up paper specific to LES conducted across multiple nodes of top-tier and exascale high-performance computers. This paper meanwhile discusses Cleo's fundamental computational design and it's performance on shared-memory architectures (single nodes). In Bayley et al. (2025), the numerics of Cleo's warm-cloud microphysics is described.

Efficient memory management is a principal feature of Cleo's design for high computational performance of large simulations. We have chosen the layout of grid-boxes and superdroplets in contiguous ordered chunks of memory to try to minimise

cache loading and contention, and support data locality. Whilst keeping superdroplets ordered requires a costly sorting algorithm, this expense will become less significant with larger simulations involving MPI domain decomposition, more complex microphysics and a fluid-dynamical core. The memory layout we have chosen also favours data transfer over SIMD operations, as is deemed appropriate for simulations involving frequent transport of a large number of superdroplets which have a low number of attributes. Since simulations of many superdroplets are exceptionally memory demanding, Cleo's optional features, such as superdroplet tracking, can be optimised out of simulations. Data output is also designed economically by following the Zarr storage specification to write binary files and by using ragged arrays to represent superdroplet data.

Cleo's resource allocation is designed flexibly so that large simulations can be economised. To exploit parallelism on HPCs, we use MPI for domain decomposition and Kokkos for abstracted and hierarchical thread parallelism. Kokkos makes Cleo concise and maintainable, but most importantly performance portable, meaning Cleo can make use of the various forms of thread-parallelism available on a diverse set of computer architectures, including those with GPUs. Further flexibility is built into how Cleo couples to a host dynamical driver. The coupling allows SDM and the dynamics to have independent domain decompositions, meaning we have greater freedom to allocate computational resources in a way that achieves load balancing and therefore produces more economical simulations.

Cleo is highly configurable as well as computationally performant. Monoids, which we define through template meta-programming with C++20 concepts, enable adaptive time-stepping and make choices regarding which variables to output and which microphysical processes to include highly flexible. In contrast with flexibility obtained through conditional code branching, flexibility via monoids has faster run-time and makes the code more readable. The coupling between Cleo and a host dynamical driver not only allows different domain decompositions, but also makes it easy to switch host model to conduct sensitivity studies and optimise the numerical methods of SDM and the dynamics-solver separately. Additionally, Cleo has flexibility over the choice of grid for SDM. All this flexibility makes Cleo well-suited for detailed study of microphysics, for example evaporation, its influence on hydrometeor evolution and atmospheric dynamics.

Cleo's current state of development raises opportunity for further optimisation and larger simulations. The bottlenecks of Cleo's algorithms for microphysics on a single node are the shuffling during SDM collisions (acute in GPU simulations), and the sorting during superdroplet motion (in CPU-only simulations). Using a different shuffling algorithm may be more optimal for GPUs, but optimisation could also come from making parts of the shuffling algorithm execute concurrently with other computations at each time-step. Similar optimisations could provide better performance of the sorting algorithm on CPUs, as well as lower its memory consumption. However, whether such optimisations are worthwhile should be assessed with more ambitious simulations, where the transport of superdroplets between nodes, additional microphysics, and/or the expense of the dynamics-solver and communication with it, could surpass these algorithms as the major bottlenecks. Given Cleo's current state of development however, we can speculate about the minimum cost of such ambitious simulations. Consider a LES with a $150\,\mathrm{km} \times 150\,\mathrm{km} \times 7\,\mathrm{km}$ domain with grid-spacing $150\,\mathrm{m}$ horizontally, $40\,\mathrm{m}$ vertically and 128 superdroplets per grid-box. In other words, $\mathcal{O}(10^{10})$ superdroplets overall. For a conservative estimate on the cost, let us assume that the overhead from particle transport via MPI and communication with the dynamics-solver increases Cleo's wall-clock by a factor of 10. Then, by using 1000 NVIDIA A100 GPUs (250 "Levante-like" GPU-nodes), the SDM setup from the performance tests presented

here suggest that $\mathcal{O}(10^7)$ superdroplets per MPI process would need approximately one second of wall-clock time for every second of simulated time and have an order of magnitude less memory consumption than the maximum on each node. That would make Cleo's throughput approximately a factor of 10 less than the strong-scaling limit of ICON (Klocke et al., 2025), and even for an extremely conservative estimate, where Cleo's performance is slower by a factor of 100, a simulation of several hours that takes several weeks to run is still conceivable. Future development of Cleo is therefore a promising avenue to pursue towards using SDM to study the interactions between cloud microphysics and mesoscale cloud organisation.

Code and data availability. The current version of Cleo is available from its GitHub page: https://github.com/yoctoyotta1024/CLEO under BSD 3-Clause license, alongside its documentation: https://yoctoyotta1024.github.io/CLEO. Version v0.39.0 is described and tested in this paper and all the code, including this Cleo version, as well as the results included this paper are archived in the dataset on Edmond under Bayley (2025). The core dependencies of Cleo v0.39.0 are: a GCC or Intel compiler for C++20, CMake, Kokkos, and yaml-cpp.

Author contributions. CJAB is the creator of Cleo and main developer, she also wrote and edited the manuscript. TK wrote parts of Cleo's code and heavily influenced its development through teaching and discussing with CJAB. Most notably TK came up with the idea for how to use C++20 concepts to create Cleo's Monoids, and the adaptive timestepping algorithm; he also introduced using the Zarr storage specification version 2 and ragged arrays for superdroplets for data output. AKN and RV supervised the project, providing direction and support, and teaching about parts of cloud physics relevant to this paper. BS conceptualised the project, oversaw Cleo's development and had many discussions with CJAB which shaped the writing of the paper and provided scientific and technical support. He also helped analyse the performance results. TK, AKN, RV, and BS all gave extensive feedback which contributed to the writing of the manuscript.

Competing interests. The contact author has declared that none of the authors has any competing interests.

Acknowledgements. C. J.A. Bayley thanks Shin-ichiro Shima (University of Hyogo, Japan) for his dedicated supervision, constructive feedback on the initial version of the manuscript, and for teaching about SDM. We gratefully acknowledge code contributions to Cleo from Sergey Kosukhin and Lukas Kluft (Max Planck Institute for Meteorology, Germany; MPI-M). A special thanks is given to Yvonne Schrader (MPI-M) for her excellent advice on the graphic designs in this paper, as well as Marco Giorgetta (MPI-M) for conducting the MPI-M internal review. We also thank the two anonymous referees of this manuscript, whose constructive comments helped to improve both its clarity and content.

A. K. Naumann and R. Vogel have received funding which supported this work from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2037 "Climate, Climatic Change, and Society" (project number 390683824). R. Vogel further acknowledges support from an ERC starting grant (ROTOR, grant no. 101116282). This project has received funding from Horizon Europe programme under Grant Agreement No 101137680 via project CERTAINTY (Cloud-aERosol inTeractions & their impActs IN The earth sYstem). The authors further express their appreciation for the work of the developers of the free and open-source

software which underlies Cleo, especially from the developers of Git, GitHub, Python, the C++ standard libraries, and, above all, Kokkos.
590 We also thank the Gesellschaft für wissenschaftliche Datenverarbetitung mbH Göttingen (GWDG) from the information and communication services Cleo's development has benefited from, and finally we thank the Deutsche Klimarechenzentrum (DKRZ) for the computer facilities from project 1183 we used to conduct this work.

References

Abel, S. J. and Shipway, B. J.: A comparison of cloud-resolving model simulations of trade wind cumulus with aircraft observations taken during RICO, Quarterly Journal of the Royal Meteorological Society, 133, 781–794, https://doi.org/10.1002/qj.55, 2007.

Ackerman, A. S., vanZanten, M. C., Stevens, B., Savic-Jovcic, V., Bretherton, C. S., Chlond, A., Golaz, J.-C., Jiang, H., Khairoutdinov, M., Krueger, S. K., Lewellen, D. C., Lock, A., Moeng, C.-H., Nakamura, K., Petters, M. D., Snider, J. R., Weinbrecht, S., and Zulauf, M.: Large-Eddy Simulations of a Drizzling, Stratocumulus-Topped Marine Boundary Layer, Monthly Weather Review, 137, 1083 – 1110, https://doi.org/10.1175/2008MWR2582.1, 2009.

Andrejczuk, M., Grabowski, W. W., Reisner, J., and Gadian, A.: Cloud-aerosol interactions for boundary layer stratocumulus in the Lagrangian Cloud Model, Journal of Geophysical Research: Atmospheres, 115, https://doi.org/10.1029/2010JD014248, 2010.

Arabas, S., Jaruga, A., Pawlowska, H., and Grabowski, W. W.: libcloudph++ 1.0: a single-moment bulk, double-moment bulk, and particle-based warm-rain microphysics library in C++, Geoscientific Model Development, 8, 1677–1707, https://doi.org/10.5194/gmd-8-1677-2015, 2015.

Bao, J. and Windmiller, J. M.: Impact of Microphysics on Tropical Precipitation Extremes in a Global Storm-Resolving Model, Geophysical Research Letters, 48, https://doi.org/10.1029/2021GL094206, 2021.

Barnes, G. M. and Garstang, M.: Subcloud Layer Energetics of Precipitating Convection, Monthly Weather Review, 110, 102 – 117, https://doi.org/10.1175/1520-0493(1982)110<0102:SLEOPC>2.0.CO;2, 1982.

Bartman, P. and Arabas, S.: On the Design of Monte-Carlo Particle Coagulation Solver Interface: A CPU/GPU Super-Droplet Method Case Study with PySDM, in: Computational Science – ICCS 2021, edited by Paszynski, M., Kranzlmüller, D., Krzhizhanovskaya, V. V., Dongarra, J. J., and Sloot, P. M. A., pp. 16–30, Springer International Publishing, Cham, ISBN 978-3-030-77964-1, 2021.

Bartman, P., Bulenok, O., Górski, K., Jaruga, A., Łazarski, G., Olesik, M. A., Piasecki, B., Singer, C. E., Talar, A., and Arabas, S.: PySDM v1: particle-based cloud modeling package for warm-rain microphysics and aqueous chemistry, Journal of Open Source Software, 7, 3219, https://doi.org/10.21105/joss.03219, 2022.

Bayley, C.: CLEO: The Fundamental Design for High Computational Performance of a New Superdroplet Model [Dataset], Edmond, https://doi.org/10.17617/3.LNRKSJ, 2025.

Bayley, C. J. A., Naumann, A. K., Poydenot, F., Vogel, R., Stevens, B., and Shima, S.-I.: CLEO: The Numerical Methods of a New Superdroplet Model including a Droplet Breakup Algorithm, EGUsphere, 2025, 1–26, https://doi.org/10.5194/egusphere-2025-4399, 2025.

Bony, S., Schulz, H., Vial, J., and Stevens, B.: Sugar, Gravel, Fish, and Flowers: Dependence of Mesoscale Patterns of Trade-Wind Clouds on Environmental Conditions, Geophysical Research Letters, 47, https://doi.org/10.1029/2019GL085988, 2020.

Brdar, S. and Seifert, A.: McSnow: A Monte-Carlo Particle Model for Riming and Aggregation of Ice Particles in a Multidimensional Microphysical Phase Space, Journal of Advances in Modeling Earth Systems, 10, 187–206, https://doi.org/10.1002/2017MS001167, 2018.

Chandrakar, K. K., Grabowski, W. W., Morrison, H., and Bryan, G. H.: Impact of Entrainment Mixing and Turbulent Fluctuations on Droplet Size Distributions in a Cumulus Cloud: An Investigation Using Lagrangian Microphysics with a Subgrid-Scale Model, Journal of the Atmospheric Sciences, 78, 2983 – 3005, https://doi.org/10.1175/JAS-D-20-0281.1, 2021.

Chandrakar, K. K., Morrison, H., Grabowski, W. W., and Lawson, R. P.: Are turbulence effects on droplet collision–coalescence a key to understanding observed rain formation in clouds?, Proceedings of the National Academy of Sciences, 121, https://doi.org/10.1073/pnas.2319664121, 2024.

de Jong, E. K., Singer, C. E., Azimi, S., Bartman, P., Bulenok, O., Derlatka, K., Dula, I., Jaruga, A., Mackay, J. B., Ward, R. X., and Arabas, S.: New developments in PySDM and PySDM-examples v2: collisional breakup, immersion freezing, dry aerosol initialization, and adaptive time-stepping, Journal of Open Source Software, 8, 4968, https://doi.org/10.21105/joss.04968, 2023.

Dziekan, P. and Zmijewski, P.: University of Warsaw Lagrangian Cloud Model (UWLCM) 2.0: adaptation of a mixed Eulerian–Lagrangian numerical model for heterogeneous computing clusters, Geoscientific Model Development, 15, 4489–4501, https://doi.org/10.5194/gmd-15-4489-2022, 2022.

Dziekan, P., Waruszewski, M., and Pawlowska, H.: University of Warsaw Lagrangian Cloud Model (UWLCM) 1.0: a modern large-eddy simulation tool for warm cloud modeling with Lagrangian microphysics, Geoscientific Model Development, 12, 2587–2606, https://doi.org/10.5194/gmd-12-2587-2019, 2019.

Edwards, H. C., Trott, C. R., and Sunderland, D.: Kokkos: Enabling manycore performance portability through polymorphic memory access patterns, Journal of Parallel and Distributed Computing, 74, 3202 – 3216, https://doi.org/10.1016/j.jpdc.2014.07.003, domain-Specific Languages and High-Level Frameworks for High-Performance Computing, 2014.

Grabowski, W. W. and Thomas, L.: Cloud droplet diffusional growth in homogeneous isotropic turbulence: bin microphysics versus Lagrangian super-droplet simulations, Atmospheric Chemistry and Physics, 21, 4059–4077, https://doi.org/10.5194/acp-21-4059-2021, 2021.

Grabowski, W. W., Morrison, H., Shima, S.-I., Abade, G. C., Dziekan, P., and Pawlowska, H.: Modeling of Cloud Microphysics: Can We Do Better?, Bulletin of the American Meteorological Society, 100, 655 – 672, https://doi.org/10.1175/BAMS-D-18-0005.1, 2019.

Hanke, M., Redler, R., Holfeld, T., and Yastremsky, M.: YAC 1.2.0: new aspects for coupling software in Earth system modelling, Geoscientific Model Development, 9, 2755–2769, https://doi.org/10.5194/gmd-9-2755-2016, 2016.

Hoffmann, F.: The Effect of Spurious Cloud Edge Supersaturations in Lagrangian Cloud Models: An Analytical and Numerical Study, Monthly Weather Review, 144, 107 – 118, https://doi.org/10.1175/MWR-D-15-0234.1, 2016.

Hoffmann, F., Raasch, S., and Noh, Y.: Entrainment of aerosols and their activation in a shallow cumulus cloud studied with a coupled LCM–LES approach, Atmospheric Research, 156, 43–57, https://doi.org/10.1016/j.atmosres.2014.12.008, 2015.

Hohenegger, C., Korn, P., Linardakis, L., Redler, R., Schnur, R., Adamidis, P., Bao, J., Bastin, S., Behravesh, M., Bergemann, M., Biercamp, J., Bockelmann, H., Brokopf, R., Brüggemann, N., Casaroli, L., Chegini, F., Datseris, G., Esch, M., George, G., Giorgetta, M., Gutjahr, O., Haak, H., Hanke, M., Ilyina, T., Jahns, T., Jungclaus, J., Kern, M., Klocke, D., Kluft, L., Kölling, T., Kornblueh, L., Kosukhin, S., Kroll, C., Lee, J., Mauritsen, T., Mehlmann, C., Mieslinger, T., Naumann, A. K., Paccini, L., Peinado, A., Praturi, D. S., Putrasahan, D., Rast, S., Riddick, T., Roeber, N., Schmidt, H., Schulzweida, U., Schütte, F., Segura, H., Shevchenko, R., Singh, V., Specht, M., Stephan, C. C., von Storch, J.-S., Vogel, R., Wengel, C., Winkler, M., Ziemen, F., Marotzke, J., and Stevens, B.: ICON-Sapphire: simulating the components of the Earth system and their interactions at kilometer and subkilometer scales, Geoscientific Model Development, 16, 779–811, https://doi.org/10.5194/gmd-16-779-2023, 2023.

Jaruga, A. and Pawlowska, H.: libcloudph++ 2.0: aqueous-phase chemistry extension of the particle-based cloud microphysics scheme, Geoscientific Model Development, 11, 3623–3645, https://doi.org/10.5194/gmd-11-3623-2018, 2018.

Jian, B., Li, J., Wang, G., Zhao, Y., Li, Y., Wang, J., Zhang, M., and Huang, J.: Evaluation of the CMIP6 marine subtropical stratocumulus cloud albedo and its controlling factors, Atmospheric Chemistry and Physics, 21, 9809–9828, https://doi.org/10.5194/acp-21-9809-2021, 2021.

Khain, A. P., Beheng, K. D., Heymsfield, A., Korolev, A., Krichak, S. O., Levin, Z., Pinsky, M., Phillips, V., Prabhakaran, T., Teller, A., van den Heever, S. C., and Yano, J.-I.: Representation of microphysical processes in cloud-resolving models: Spectral (bin) microphysics versus bulk parameterization, Reviews of Geophysics, 53, 247–322, https://doi.org/10.1002/2014RG000468, 2015.

Klocke, D., Frauen, C., Engels, J. F., Alexeev, D., Redler, R., Schnur, R., Haak, H., Kornblueh, L., Brüggemann, N., Chegini, F., Römmer, M., Hoffmann, L., Griessbach, S., Bode, M., Coles, J., Gila, M., Sawyer, W., Calotoiu, A., Budanaz, Y., Mazumder, P., Copik, M., Weber, B., Herten, A., Bockelmann, H., Hoefler, T., Hohenegger, C., and Stevens, B.: Computing the Full Earth System at 1km Resolution, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '25, p. 125–136, Association for Computing Machinery, New York, NY, USA, ISBN 9798400714665, https://doi.org/10.1145/3712285.3771789, 2025.

La, I., Grabowski, W. W., Kim, Y., Kim, S., and Yum, S. S.: Lagrangian Particle–Based Simulation of Aerosol-Dependent Vertical Variation of Cloud Microphysics in a Laboratory Convection Cloud Chamber, EGUsphere, 2025, 1–32, https://doi.org/10.5194/egusphere-2025-3952, 2025.

Lang, T., Naumann, A. K., Buehler, S. A., Stevens, B., Schmidt, H., and Aemisegger, F.: Sources of Uncertainty in Mid-Tropospheric Tropical Humidity in Global Storm-Resolving Simulations, Journal of Advances in Modeling Earth Systems, 15, https://doi.org/10.1029/2022MS003443, 2023.

Li, X.-Y., Brandenburg, A., Svensson, G., Haugen, N. E. L., Mehlig, B., and Rogachevskii, I.: Effect of Turbulence on Collisional Growth of Cloud Droplets, Journal of the Atmospheric Sciences, 75, 3469 – 3487, https://doi.org/10.1175/JAS-D-18-0081.1, 2018.

Li, X.-Y., Brandenburg, A., Svensson, G., Haugen, N. E. L., Mehlig, B., and Rogachevskii, I.: Condensational and Collisional Growth of Cloud Droplets in a Turbulent Environment, Journal of the Atmospheric Sciences, 77, 337 – 353, https://doi.org/10.1175/JAS-D-19-0107.1, 2020.

Matsushima, T., Nishizawa, S., and Shima, S.: Overcoming computational challenges to realize meter- to submeter-scale resolution in cloud simulations using the super-droplet method, Geoscientific Model Development, 16, 6211–6245, https://doi.org/10.5194/gmd-16-6211-2023, 2023.

Miyakawa, T., Satoh, M., Miura, H., Tomita, H., Yashiro, H., Noda, A. T., Yamada, Y., Kodama, C., Kimoto, M., and Yoneyama, K.: Madden–Julian Oscillation prediction skill of a new-generation global model demonstrated using a supercomputer, Nature Communications, 5, 3769, https://doi.org/10.1038/ncomms4769, 2014.

Morrison, H., Witte, M., Bryan, G. H., Harrington, J. Y., and Lebo, Z. J.: Broadening of Modeled Cloud Droplet Spectra Using Bin Microphysics in an Eulerian Spatial Domain, Journal of the Atmospheric Sciences, 75, 4005 – 4030, https://doi.org/10.1175/JAS-D-18-0055.1, 2018.

Morrison, H., van Lier-Walqui, M., Fridlind, A. M., Grabowski, W. W., Harrington, J. Y., Hoose, C., Korolev, A., Kumjian, M. R., Milbrandt, J. A., Pawlowska, H., Posselt, D. J., Prat, O. P., Reimel, K. J., Shima, S.-I., van Diedenhoven, B., and Xue, L.: Confronting the Challenge of Modeling Cloud and Precipitation Microphysics, Journal of Advances in Modeling Earth Systems, 12, https://doi.org/10.1029/2019MS001689, 2020.

Naumann, A. K. and Seifert, A.: A Lagrangian drop model to study warm rain microphysical processes in shallow cumulus, Journal of Advances in Modeling Earth Systems, 7, 1136–1154, https://doi.org/10.1002/2015MS000456, 2015.

Naumann, A. K., Esch, M., and Stevens, B.: How the representation of microphysical processes affects tropical condensate in the global storm-resolving model ICON, Atmospheric Chemistry and Physics, 25, 6429–6444, https://doi.org/10.5194/acp-25-6429-2025, 2025.

Nishizawa, S., Yashiro, H., Sato, Y., Miyamoto, Y., and Tomita, H.: Influence of grid aspect ratio on planetary boundary layer turbulence in large-eddy simulations, Geoscientific Model Development, 8, 3393–3419, https://doi.org/10.5194/gmd-8-3393-2015, 2015.

Nuijens, L. and Siebesma, A. P.: Boundary Layer Clouds and Convection over Subtropical Oceans in our Current and in a Warmer Climate, Current Climate Change Reports, 5, 80–94, 2019.

Nuijens, L., Stevens, B., and Siebesma, A. P.: The Environment of Precipitating Shallow Cumulus Convection, Journal of the Atmospheric Sciences, 66, 1962 – 1979, https://doi.org/10.1175/2008JAS2841.1, 2009.

Radtke, J., Naumann, A. K., Hagen, M., and Ament, F.: The relationship between precipitation and its spatial pattern in the trades observed during EUREC4A, Quarterly Journal of the Royal Meteorological Society, 148, 1913–1928, https://doi.org/10.1002/qj.4284, 2022.

Radtke, J., Vogel, R., Ament, F., and Naumann, A. K.: Spatial Organisation Affects the Pathway to Precipitation in Simulated Trade-Wind Convection, Geophysical Research Letters, 50, https://doi.org/10.1029/2023GL103579, 2023.

Randall, D., Khairoutdinov, M., Arakawa, A., and Grabowski, W.: Breaking the Cloud Parameterization Deadlock, Bulletin of the American Meteorological Society, 84, 1547 – 1564, https://doi.org/10.1175/BAMS-84-11-1547, 2003.

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

Rogers, R. R., Baumgardner, D., Ethier, S. A., Carter, D. A., and Ecklund, W. L.: Comparison of Raindrop Size Distributions Measured by Radar Wind Profiler and by Airplane, Journal of Applied Meteorology and Climatology, 32, 694 – 699, https://doi.org/10.1175/1520-0450(1993)032<0694:CORSDM>2.0.CO;2, 1993.

Sato, Y., Nishizawa, S., Yashiro, H., et al.: Impacts of cloud microphysics on trade wind cumulus: which cloud microphysics processes contribute to the diversity in a large eddy simulation?, Progress in Earth and Planetary Science, 2, 23, https://doi.org/10.1186/s40645-015-0053-6, 2015.

Sato, Y., Shima, S.-i., and Tomita, H.: A grid refinement study of trade wind cumuli simulated by a Lagrangian cloud microphysical model: the super-droplet method, Atmospheric Science Letters, 18, 359–365, https://doi.org/https://doi.org/10.1002/asl.764, 2017.

Schulz, H. and Stevens, B.: Evaluating Large-Domain, Hecto-Meter, Large-Eddy Simulations of Trade-Wind Clouds Using EUREC4A Data, Journal of Advances in Modeling Earth Systems, 15, https://doi.org/10.1029/2023MS003648, 2023.

Schulz, H., Eastman, R., and Stevens, B.: Characterization and Evolution of Organized Shallow Convection in the Downstream North Atlantic Trades, Journal of Geophysical Research: Atmospheres, 126, https://doi.org/10.1029/2021JD034575, 2021.

Seifert, A.: On the Parameterization of Evaporation of Raindrops as Simulated by a One-Dimensional Rainshaft Model, Journal of the Atmospheric Sciences, 65, 3608 – 3619, https://doi.org/10.1175/2008JAS2586.1, 2008.

Seifert, A. and Heus, T.: Large-eddy simulation of organized precipitating trade wind cumulus clouds, Atmospheric Chemistry and Physics, 13, 5631–5645, https://doi.org/10.5194/acp-13-5631-2013, 2013.

Shima, S., Kusano, K., Kawano, A., Sugiyama, T., and Kawahara, S.: The super-droplet method for the numerical simulation of clouds and precipitation: a particle-based and probabilistic microphysics model coupled with a non-hydrostatic model, Quarterly Journal of the Royal Meteorological Society, 135, 1307–1320, https://doi.org/10.1002/qj.441, 2009.

Shima, S., Sato, Y., Hashimoto, A., and Misumi, R.: Predicting the morphology of ice particles in deep convection using the super-droplet method: development and evaluation of SCALE-SDM 0.2.5-2.2.0, -2.2.1, and -2.2.2, Geoscientific Model Development, 13, 4107–4157, https://doi.org/10.5194/gmd-13-4107-2020, 2020.

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

Stevens, B. and Seifert, A.: Understanding macrophysical outcomes of microphysical choices in simulations of shallow cumulus convection, Journal of the Meteorological Society of Japan. Ser. II, 86A, 143–162, https://doi.org/10.2151/jmsj.86A.143, 2008.

Stevens, B., Walko, R. L., Cotton, W. R., and Feingold, G.: The Spurious Production of Cloud-Edge Supersaturations by Eulerian Models, Monthly Weather Review, 124, 1034 – 1041, https://doi.org/10.1175/1520-0493(1996)124<1034:TSPOCE>2.0.CO;2, 1996.

Stevens, B., Vali, G., Comstock, K., Wood, R., van Zanten, M. C., Austin, P. H., Bretherton, C. S., and Lenschow, D. H.: POCKETS OF OPEN CELLS AND DRIZZLE IN MARINE STRATOCUMULUS, Bulletin of the American Meteorological Society, 86, 51 – 58, https://doi.org/10.1175/BAMS-86-1-51, 2005.

Stevens, B., Acquistaoace, C., Hansen, A., Heinze, R., Klinger, C., Klocke, D., Rybka, H., Schubotz, W., Windmiller, J., Adamidis, P., Arka, I., Barlakas, V., Biercamp, J., Brueck, M., Brune, S., Buehler, S. A., Burkhardt, U., Cioni, G., Costa-Suros, M., Crewell, S., Crüger, T., Deneke, H., Friedrichs, P., Henken, C. C., Hohenegger, C., Jacob, M., Jakub, F., Kalthoff, N., Köhler, M., van Laar, T. W., Li, P., Löhnert, U., Macke, A., Madenach, N., Mayer, B., Nam, C., Naumann, A. K., Peters, K., Poll, S., Quaas, J., Röber, N., Rochetin, N., Scheck, L., Schemann, V., Schnitt, S., Seifert, A., Senf, F., Shapkalijevski, M., Simmer, C., Singh, S., Sourdeval, O., Spickermann, D., Strandgren, J., Tessiot, O., Vercauteren, N., Vial, J., Voigt, A., and Zängl, G.: The Added Value of Large-Eddy and Storm-Resolving Models for Simulating Clouds and Precipitation, Journal of the Meteorological Society of Japan, 98, 395–435, https://doi.org/10.2151/jmsj.2020-021, 2020.

Suematsu, T., Kodama, C., Yamada, Y., Miura, H., Takasuka, D., and Miyakawa, T.: Microphysics dependency in 3.5km NICAM DYAMOND phase 2 experiments, in: AGU Fall Meeting Abstracts, vol. 2021, 2021.

Sölch, I. and Kärcher, B.: A large-eddy model for cirrus clouds with explicit aerosol and ice microphysics and Lagrangian ice particle tracking, Quarterly Journal of the Royal Meteorological Society, 136, 2074–2093, https://doi.org/10.1002/qj.689, 2010.

Takasuka, D., Kodama, C., Suematsu, T., Ohno, T., Yamada, Y., Seiki, T., Yashiro, H., Nakano, M., Miura, H., Noda, A. T., Nasuno, T., Miyakawa, T., and Masunaga, R.: How Can We Improve the Seamless Representation of Climatological Statistics and Weather Toward Reliable Global K-Scale Climate Simulations?, Journal of Advances in Modeling Earth Systems, 16, https://doi.org/10.1029/2023MS003701, 2024.

Trott, C., Berger-Vergiat, L., Poliakoff, D., Rajamanickam, S., Lebrun-Grandie, D., Madsen, J., Al Awar, N., Gligoric, M., Shipman, G., and Womeldorff, G.: The Kokkos EcoSystem: Comprehensive Performance Portability for High Performance Computing, Computing in Science Engineering, 23, 10–18, https://doi.org/10.1109/MCSE.2021.3098509, 2021.

Trott, C. R., Lebrun-Grandié, D., Arndt, D., Ciesko, J., Dang, V., Ellingwood, N., Gayatri, R., Harvey, E., Hollman, D. S., Ibanez, D., Liber, N., Madsen, J., Miles, J., Poliakoff, D., Powell, A., Rajamanickam, S., Simberg, M., Sunderland, D., Turcksin, B., and Wilke, J.: Kokkos 3: Programming Model Extensions for the Exascale Era, IEEE Transactions on Parallel and Distributed Systems, 33, 805–817, https://doi.org/10.1109/TPDS.2021.3097283, 2022.

Unterstrasser, S., Hoffmann, F., and Lerch, M.: Collection/aggregation algorithms in Lagrangian cloud microphysical models: rigorous evaluation in box model simulations, Geoscientific Model Development, 10, 1521–1548, https://doi.org/10.5194/gmd-10-1521-2017, 2017.

vanZanten, M. C., Stevens, B., Nuijens, L., Siebesma, A. P., Ackerman, A. S., Burnet, F., Cheng, A., Couvreux, F., Jiang, H., Khairoutdinov, M., Kogan, Y., Lewellen, D. C., Mechem, D., Nakamura, K., Noda, A., Shipway, B. J., Slawinska, J., Wang, S., and Wyszogrodzki, A.: Controls on precipitation and cloudiness in simulations of trade-wind cumulus as observed during RICO, Journal of Advances in Modeling Earth Systems, 3, https://doi.org/10.1029/2011MS000056, 2011.

Vogel, R., Konow, H., Schulz, H., and Zuidema, P.: A climatology of trade-wind cumulus cold pools and their link to mesoscale cloud organization, Atmospheric Chemistry and Physics, 21, 16 609–16 630, https://doi.org/10.5194/acp-21-16609-2021, 2021.

Yin, C., Shima, S., Xue, L., and Lu, C.: Simulation of marine stratocumulus using the super-droplet method: numerical convergence and comparison to a double-moment bulk scheme using SCALE-SDM 5.2.6-2.3.1, Geoscientific Model Development, 17, 5167–5189, https://doi.org/10.5194/gmd-17-5167-2024, 2024.

Listing 5 Example code showing how observers are combined in Cleo. Here we choose "$\gg$" to be the operator in the definition of $A \oplus B = C$ for observers. Monitors are also monoid (also with their own C++20 concept, associative operation, and identity element) but we omit their definition here for brevity.

```
/* define how to combine observers */
template <Observer Obs1, Observer Obs2, Monitor Mo>
struct CombinedObserver {
 private:
  Obs1 a; /**< First Observer. */
  Obs2 b; /**< Second Observer. */
  Mo mo;  /**< Combination of First and Second Observers' Monitors */

 public:
  void before_timestepping(const view_constgbx gbxs) const {
    a.before_timestepping(gbxs);
    b.before_timestepping(gbxs);
  }

  void after_timestepping() const {
    a.after_timestepping();
    b.after_timestepping();
  }

  unsigned int next_step(const unsigned int t) const {
    return Kokkos::min(a.next_step(t), b.next_step(t));
  }

  bool on_step(const unsigned int t) const {
    return a.on_step(t) || b.on_step(t);
  }

  void at_start_step(const unsigned int t, const view_constgbx,
                     const view_constsupers supers) const {
    a.at_start_step(t, gbxs, supers);
    b.at_start_step(t, gbxs, supers);
  }

  Monitor auto get_monitor() const { return mo; }
};

auto operator>>(const Observer auto obs1, const Observer auto obs2) {
  const Monitor auto mo12 =
      CombinedMonitor{obs1.get_monitor(), obs2.get_monitor()};
  return CombinedObserver{obs1, obs2, mo12};
}

// ... //

/* use a combination of observers */
const Observer auto obs1 = StreamOutObserver(config);
const Observer auto obs2 = TimeObserver(config, dataset);
const Observer auto obs3 = StateObserver(config, dataset);
const Observer auto obs4 = SuperdropsObserver(config, dataset);
const Observer auto observer =
    obs4 >> obs3 >> obs2 >> obs1;  // change this line to change observer
}
```

Listing 6 The identity element of the observer monoid in Cleo.

```
/* define the identity element for an Observer */
struct NullObserver {
  void before_timestepping(const view_constgbx gbxs) const {}

  void after_timestepping() const {}

  unsigned int next_step(const unsigned int t) const {
    return std::numeric_limits<unsigned int>::max();
  }

  bool on_step(const unsigned int t) const { return false; }

  void at_start_step(const unsigned int t, const view_constgbx gbxs,
                     const view_constsupers supers) const {}

  Monitor auto get_monitor() const { return NullMonitor{}; }
};
```

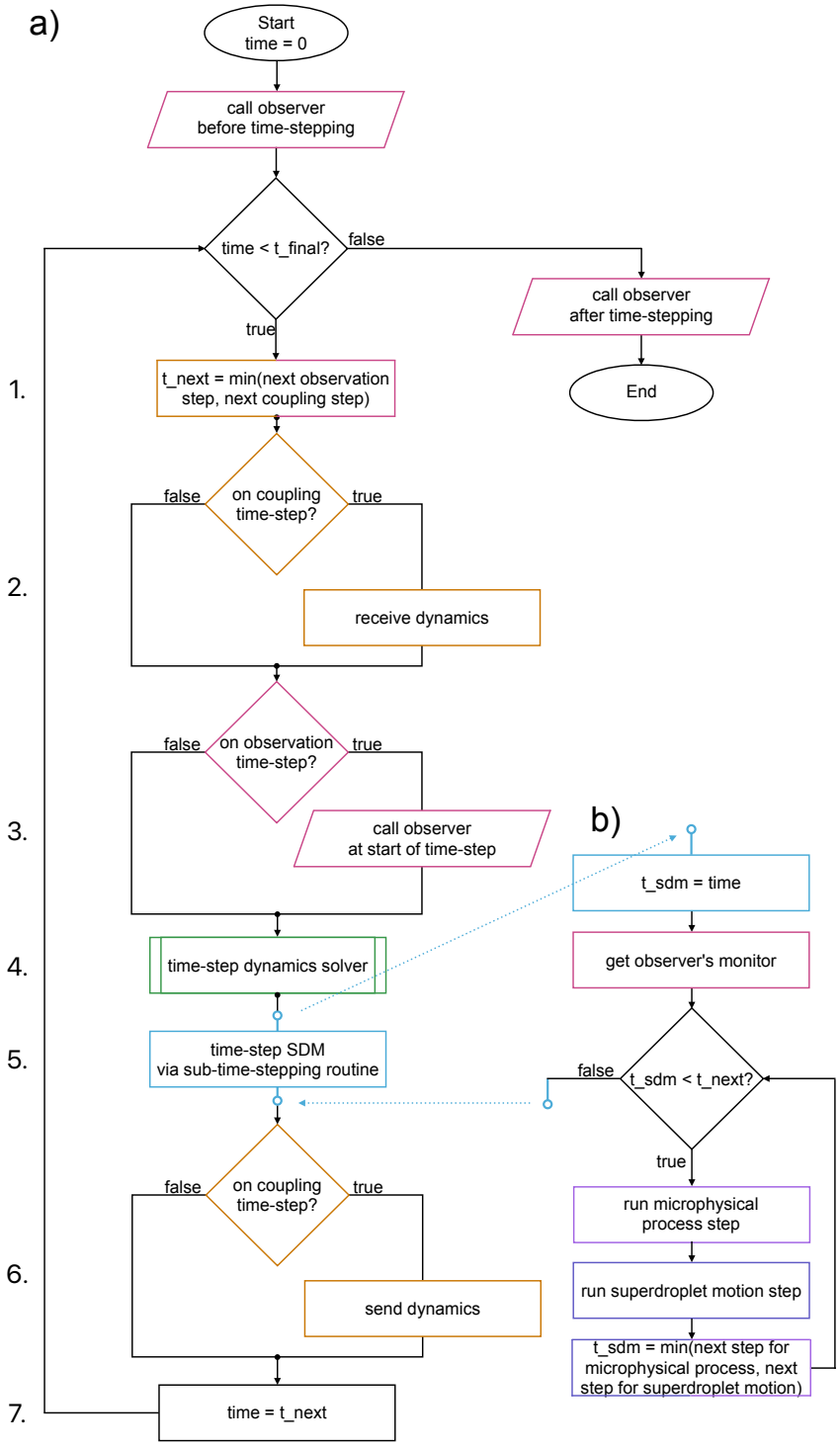

Figure 5. a) The flow diagram for Cleo's time-stepping routine. Each step is coloured by which of Cleo's structures it involves. The steps involved in the SDM sub-time-stepping routine are shown in b) and are an extension of the logic presented in Figure 4a to include output and superdroplet motion as well as microphysics.

	CPU-only	CUDA-enabled[1]
CPUs	2 AMD 7763	2 AMD 7763
main memory /GB	256	512
number of threads (cores)	256 (128)	256 (128)
cores in (L3 cache, NUMA domain, single socket)	(8, 16, 64)	(8, 16, 64)
GPUs	-	1 NVIDIA A100
main memory /GB	-	40
compiler	intel-oneapi-compilers/2023.2.1-gcc-11.2.0	gcc/11.2.0-gcc-11.2.0
MPI	openmpi@4.1.5%oneapi/3ccjsdq	openmpi@4.1.2%gcc@11.2.0
CMake	cmake@3.23.1%oneapi	cmake@3.26.3%gcc@=11.2.0/fuvwuhz
CUDA		cuda@12.2.0%gcc@=11.2.0

Table 1. Configuration and software of Levante HPC for the performance tests of Cleo on a single node with only CPUs (for Serial, OpenMP, and C++Threads builds), and with CUDA enabled.

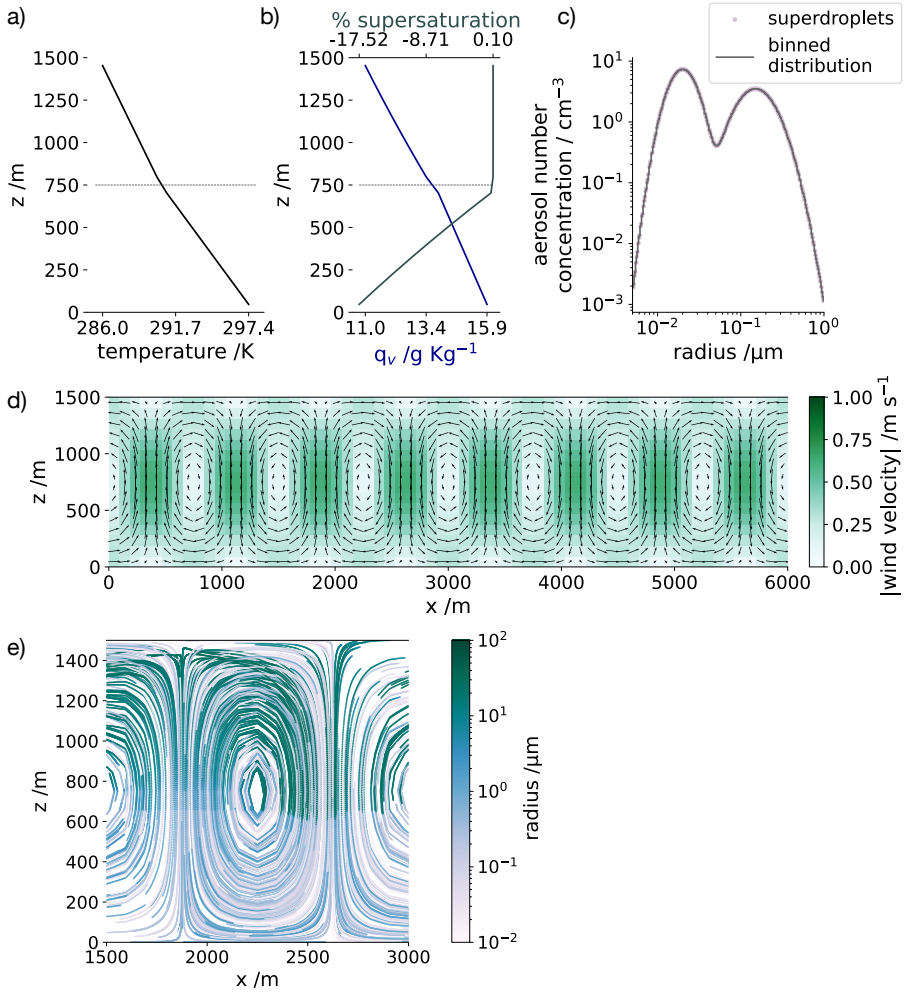

Figure 6. An example of the performance tests with 16384 grid-boxes. (a-d) The initial conditions, (e) The growth and trajectories of a random sample of 500 superdroplets between $1500\,\mathrm{m} < x < 3000\,\mathrm{m}$ up-to $80\,\mathrm{mins}$ for the test case with 4.194×10^6 superdroplets overall.

Grid-Boxes	Total Superdroplets	Δx / m	Δy / m	Δz / m
1	2.560×10^2	6000.0	300.0	1500.0
8	2.048×10^3	1500.0	150.0	1500.0
64	1.638×10^4	750.0	75.0	750.0
512	1.311×10^5	375.0	37.5	375.0
2048	5.243×10^5	187.5	37.5	187.5
4096	1.049×10^6	187.5	18.8	187.5
16384	4.194×10^6	93.8	18.8	93.8
32768	8.389×10^6	93.8	9.4	93.8
131072	3.355×10^7	46.9	9.4	46.9

Table 2. The problem size for the performance tests in terms of the number of grid-boxes composing the domain and the consequent total number of superdroplets and domain grid-spacing.

	Time-step / s	Additional Configuration Notes
droplet motion	3	terminal velocity parametrisation from Rogers et al. (1993)
condensation/evaporation	1 (default)	minimum sub-time-step: 0.001 s
		absolute and relative tolerances: 0.01 and 0.0
collision-coalescence	1	hydrodynamic kernel from Simmel et al. (2002)

Table 3. SDM configuration for the tests of Cleo's superdroplet- and strong-scaling.

	CUDA Speed-Up		
#SDs in Domain	SDM	Microphysics	Motion
1.638×10^4	10.9	12.4	6.86
1.311×10^5	49.3	50.8	43.8
1.049×10^6	76.5	69.9	139.5
3.355×10^7	79.0	71.6	152.3

Table 4. The speed-up of wall-clock time for the CUDA build relative to the serial simulations for problem sizes plotted in Figure 8.

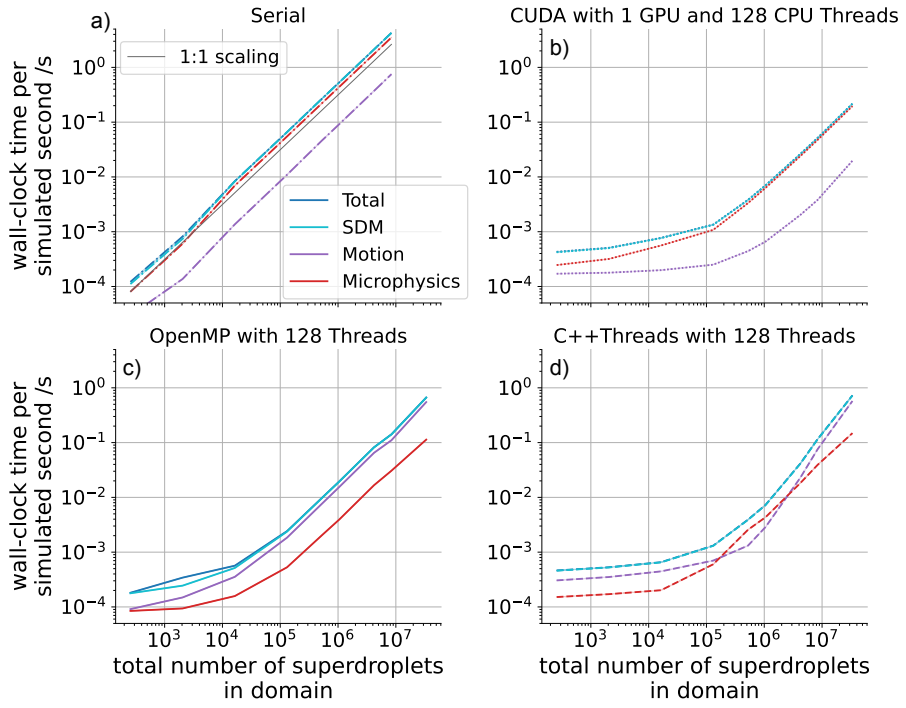

Figure 7. The superdroplet-scaling of the wall-clock time for each simulated second given the maximum set of resources for each build on Levante HPC: a) Serial (dash-dotted), b) CUDA (dotted), c) OpenMP (solid), and d) C++Threads (dashed). The colours decompose the total time (blue) into the time spent on SDM (cyan), motion (purple) and microphysics (red)

	Serial	OpenMP	C++Threads	CUDA
% of high-water memory consumption				
Superdroplets	99.3	97.5	97.5	99.1
Grid-Boxes	0.2	0.2	0.2	0.6
Apparent size / bytes				
Superdroplet	314.4	309.8	308.6	332.9
Grid-Box	162.1	162.7	162.0	515.9

Table 5. The apparent size of superdroplets and grid-boxes calculated from their contribution to the high-water memory consumption for the simulations with 32768 grid-boxes (256 superdroplets per grid-box).

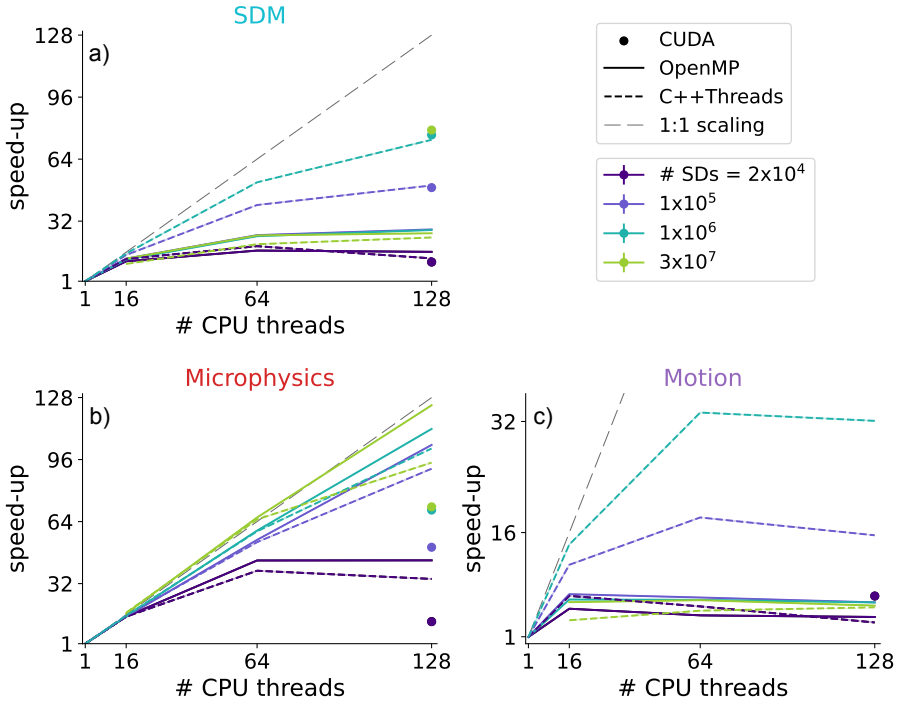

Figure 8. The strong-scaling of the speed-up of wall-clock time relative to the serial simulations with increasing number of CPU threads on Levante HPC. The line-style denotes the build and the colours from purple to green indicate increasing problem size in terms of the total number of superdroplets in the domain. The speed-up of motion from CUDA exceeds the y-axis scale and is reported in Table 4 instead.

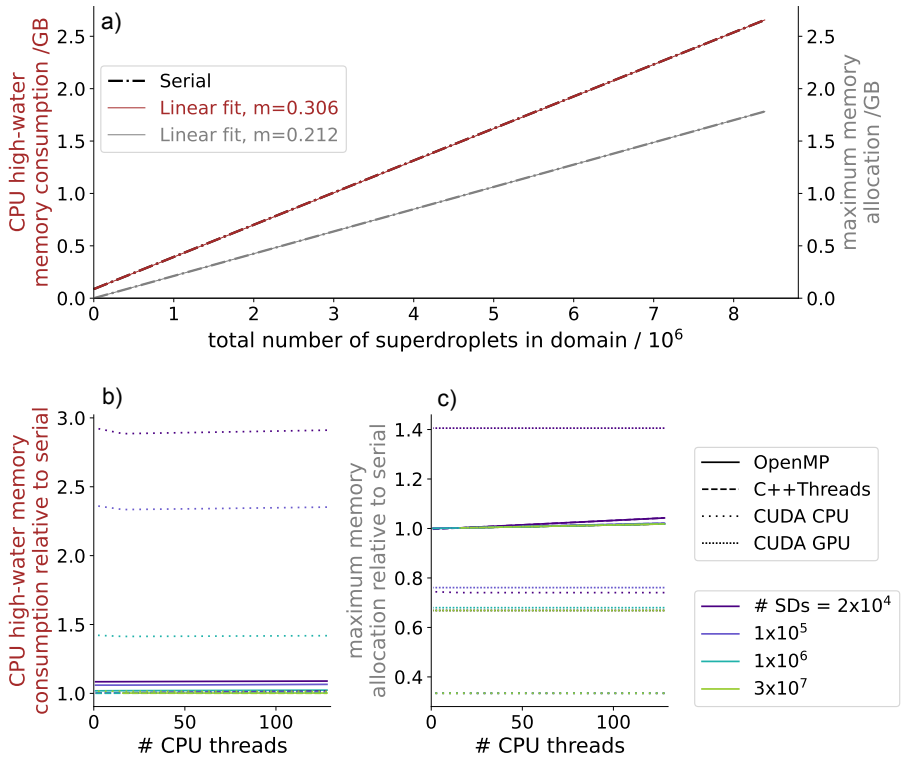

Figure 9. a) The superdroplet-scaling of the serial build for the resident set size high-water memory consumption (brown) and for the maximum memory allocation (grey); b and c) The strong-scaling of the same memory footprints for the parallelised builds: OpenMP (solid), C++Threads (dashed) and CUDA (dotted). The colours from purple to green indicate increasing problem size in terms of the total number of superdroplets in the domain.