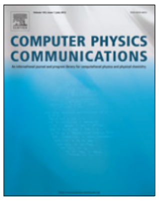
Computer Physics Communications 185 (2014) 3269–3290

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc



DNS of hydrodynamically interacting droplets in turbulent clouds: Parallel implementation and scalability analysis using 2D domain decomposition

Orlando Ayala[a,c,*], Hossein Parishani[b], Liu Chen[b], Bogdan Rosa[d], Lian-Ping Wang[b]

[a] Department of Engineering Technology, Old Dominion University, Norfolk, VA 23529, United States
[b] Department of Mechanical Engineering, University of Delaware, Newark, DE 19716-3140, United States
[c] Centro de Métodos Numéricos en Ingeniería, Escuela de Ingeniería y Ciencias Aplicadas, Universidad de Oriente, Puerto La Cruz, Venezuela, Cuba
[d] Institute of Meteorology and Water Management-National Research Institute, 61 Podlesna Street, 01-673 Warsaw, Poland

ARTICLE INFO

Article history:
Received 18 September 2013
Received in revised form
6 September 2014
Accepted 9 September 2014
Available online 21 September 2014

Keywords:
Parallel computing
Particle collision
Hydrodynamic interaction
Homogeneous isotropic turbulence
Complexity analysis
Particle-laden flows

ABSTRACT

The study of turbulent collision of cloud droplets requires simultaneous considerations of the transport by background air turbulence (i.e., geometric collision rate) and influence of droplet disturbance flows (i.e., collision efficiency). In recent years, this multiscale problem has been addressed through a hybrid direct numerical simulation (HDNS) approach (Ayala et al., 2007). This approach, while currently is the only viable tool to quantify the effects of air turbulence on collision statistics, is computationally expensive. In order to extend the HDNS approach to higher flow Reynolds numbers, here we developed a highly scalable implementation of the approach using 2D domain decomposition. The scalability of the parallel implementation was studied using several parallel computers, at 512^3 and 1024^3 grid resolutions with $\mathcal{O}(10^6)$–$\mathcal{O}(10^7)$ droplets. It was found that the execution time scaled with number of processors almost linearly until it saturates and deteriorates due to communication latency issues.

To better understand the scalability, we developed a complexity analysis by partitioning the execution tasks into computation, communication, and data copy. Using this complexity analysis, we were able to predict the scalability performance of our parallel code. Furthermore, the theory was used to estimate the maximum number of processors below which the approximately linear scalability is sustained. We theoretically showed that we could efficiently solved problems of up to 8192^3 with $\mathcal{O}(100{,}000)$ processors. The complexity analysis revealed that the pseudo-spectral simulation of background turbulent flow for a dilute droplet suspension typical of cloud conditions typically takes about 80% of the total execution time, except when the droplets are small (less than 5 μm in a flow with energy dissipation rate of 400 cm^2/s^3 and liquid water content of 1 g/m^3), for which case the particle–particle hydrodynamic interactions become the bottleneck. The complexity analysis was also used to explore some alternative methods to handle FFT calculations within the flow simulation and to advance droplets less than 5 μm in radius, for better computational efficiency. Finally, preliminary results are reported to shed light on the Reynolds number-dependence of collision kernel of non-interacting droplets.

1. Introduction

Dispersed flows consist of a carrier fluid and at least one discrete phase such as droplets, bubbles, or particles. They are present in many industrial processes and in the environment such as pneumatic conveying systems, coal combustion, sediment transport by wind and water waves, and clouds and precipitation via rain and snow. Despite the industrial and environmental relevance, fundamental understanding of the mechanisms governing dispersed flows is incomplete due to a host of length and time scales associated with the dispersed phase and its interactions with the carrier phase. Typically, the number of variables characterizing a dispersed flow is considerably larger than that of the single-phase flow. In certain applications such as turbulent collision of dispersed particles, direct experimental measurements of both the carrier phase and the dispersed phase encompassing all relevant scales are often difficult.

* Corresponding author at: Department of Engineering Technology, Old Dominion University, Norfolk, VA 23529, United States. Tel.: +1 3023974981.
E-mail addresses: omayalah@udel.edu, omayalah@gmail.com, oayala@odu.edu (O. Ayala).

http://dx.doi.org/10.1016/j.cpc.2014.09.005
0010-4655/© 2014 Elsevier B.V. All rights reserved.

In recent years, due to advances in both computing power and algorithms, numerical simulations of turbulent dispersed flows based on first principles have been established as the third pillar for scientific discovery in addition to theory and experimentation. In the so-called point-particle direct numerical simulations, the carrier phase is solved through a set of continuum differential equations while each particle in the dispersed phase is tracked through a modeled equation of motion. This Eulerian–Lagrangian approach has been advanced now to a point that various particle–fluid, particle–particle, and particle–wall interactions could be included (*e.g.*, Elghobashi [1,2], Sommerfeld [3], Wang et al. [4]). This advantage makes the Eulerian–Lagrangian approach a major research tool for studying turbulent dispersed flow. However, this approach is computationally expensive when the number of discrete particles is large and most scales of the carrier-phase turbulent flow must be resolved.

In order to realize the full potential of the Eulerian–Lagrangian approach for dispersed flows, it is necessary to employ highly scalable computation using a large number of processors (e.g., 10,000 to 100,000) typically available on the current petascale supercomputers. The distributed-memory, many-core architecture on these computers calls for parallel implementation based on the *divide-and-conquer* approach so simulations of large problem sizes can be performed efficiently.

The Eulerian flow simulation of the carrier phase can be efficiently scaled using a domain decomposition (dd) approach, where the domain is explicitly decomposed and mapped onto the processors in such a way that a load balancing is maintained and communication between processors is minimized and well structured. Depending on whether the boundaries of the domain are moving or not, the decomposition could be dynamic or static, i.e. the domain covered by each processor could change with time or be fixed. Alfonsi [5] surveyed a compendium of parallel implementations of direct numerical simulations of turbulent flows, all based on dd using a variety of flow solvers including the pseudo spectral method [6–12], lattice Boltzmann method [13,14], finite element method [15], and finite difference method [16–19]. Due to different machine architectures in terms of memory (distributed memory, shared memory, or combination), the techniques for parallel implementation vary from OpenMP, MPI, or a hybrid approach.

There are alternative strategies for parallel implementation of the dispersed phase, and the optimal approach may depend on the nature of coupling interactions considered (i.e., one-, two-, or four-way interactions) [20,21]. Parallel implementation strategies in a multi-core machine [22,23] can be categorized into: (1) domain decomposition, (2) particle decomposition, (3) particle-interaction decomposition, and (4) replicated data method. In the domain decomposition (dd) method, the physical space is partitioned and each processor handles particles spatially located in a subdomain assigned to the processor [24–26]. The advantage of dd is that it is fairly simple to implement; however, the disadvantage is that a good load balance could be hard to achieve if particles are not evenly distributed in space. The communication of the particle data takes place between neighboring processors only. Onishi et al. [26] allowed two processors to share a halo region where the same particles were tracked. This reduces the communication need but increases computation.

In the particle decomposition, the particles are evenly distributed to different groups and each group is assigned to a processor [22]. A good load balance is achieved relatively easily but each processor has to store the same flow field data or otherwise an additional communication step is needed to make the flow field data available to a given processor.

The particle-interaction decomposition is a scheme used when particles interact among themselves, besides the fluid–particle interactions, such as in the N-body problems [27]. In this case, the pair interactions within the N-body problem are divided among all the processors. This is usually done for relatively small systems.

Lastly, in the replicated data method, particle coordinates (or other particle information) are replicated in each processor, usually through a broadcast operation. Its use is usually combined with the other schemes [20,14]. The obvious problem of this method is a poor scalability as well as the memory limitation it might impose.

In recent years, considerable attention has been paid to parallelization strategies of dispersed flow computation using the Eulerian–Lagrangian approach where the carrier flow is solved by the Reynolds-averaged Navier–Stokes equation [17,18,28,29], large-eddy simulation [20,30], and direct numerical simulation [26]. To the best of our knowledge, none was intended for large computers. In addition, there has been no systematic complexity analysis of parallel codes for dispersed flow. The few available studies on codes focused primarily on the flow simulation.

Our study is motivated by an important problem in warm-rain cloud microphysics, namely, the turbulent collision-coalescence of cloud droplets [31–35]. Here the dispersed phase is made of small water droplets.[1] It is well established that small-scale turbulent eddies down to the Kolmogorov scale affect collision statistics while large-scale turbulent eddies mix and disperse the droplets [35]. The study of turbulent collision of cloud droplets then requires direct numerical simulation of the carrier flow where all scales of undisturbed fluid motion are numerically resolved.

Under the working hypothesis that the small-scale fluid motion dominates the pair collision statistics, the Eulerian–Lagrangian approach with simulated DNS flow field was first applied to study the geometric collision of cloud droplets [36–38]. In order to study the effect of turbulence on the collision efficiency, droplet–droplet hydrodynamic interactions must also be considered. By embedding Stokes disturbance flows due to droplets, Wang et al. [39] and Ayala et al. [40] introduced the Hybrid DNS (HDNS) scheme. Since in clouds the droplet volume fraction is small, we assume one-way interaction between the dispersed phase and the background turbulent flow. The disturbance flows due to the presence of droplets are treated separately, only for the purpose to incorporate droplet–droplet hydrodynamic interactions at the sub-Kolmogorov scale of the background turbulent flow.

A key aspect of the HDNS approach is the determination of disturbance flow velocity at the location of each particle, requiring an iterative algorithm to solve a large linear system at each time step. This adds yet another level of computational complexity. The initial implementation [40] was carried out with loop-level parallelization using OpenMP, which is limited to the use of processors within a same computational node. To extend the HDNS to a larger domain (or larger flow Reynolds number), Rosa and Wang [41] considered an MPI implementation using dd in one spatial dimension (1D). Their MPI implementation did not include droplet hydrodynamic interactions.

In this paper, we report an MPI implementation of the HDNS code based on dd in two spatial dimensions (2D) for dilute droplet suspensions typical of cloud conditions. The first objective is to discuss the implementation details that lead to a highly scalable HDNS

[1] The terms "particles" and "droplets" are used interchangeably in this paper. "Particles" is a more general term for other applications for which the current study may also be applied.

code for turbulent collision of hydrodynamically-interacting cloud droplets. The second objective is to develop an extensive complexity analysis to predict the execution time. We must stress that the basic concepts invoked in our complexity analysis are not necessarily new but previous related studies on parallel implementation did not provide such an analysis in a quantitative manner. In addition, a detailed complexity analysis of such complex computational problem is not a trivial task. Therefore, our systematic complexity analysis fills a gap in the scalable computation of turbulent dispersed flow.

The paper is organized as follows. In Section 2, we review the HDNS approach. Section 3 describes in detail the parallel implementation of the approach using a 2D dd strategy. We will also develop a complexity analysis to understand the parallel performance of the code. Section 4 contains evaluation and testing results of the implementation, using several state-of-the-art supercomputers (Stampede, Yellowstone, and Kraken). Subsequently, after the complexity analysis is validated, it is applied to explore alternative treatments for some components of the code. To demonstrate the capability of the code, we discuss in Section 5 the Reynolds number dependence of the collision kernel. Finally, conclusions are summarized in Section 6.

2. Model formulation and numerical solution method

We only provide a brief description of the HDNS approach here as the basic ideas and algorithms have been presented in [40]. We consider a dilute suspension of droplets in a background turbulent air flow $\mathbf{U}(\mathbf{x}, t)$. Under one-way coupling, the background flow is solved by a pseudo-spectral method in a periodic domain, using the Navier–Stokes equation

$$\frac{\partial \mathbf{U}}{\partial t} = \mathbf{U} \times \boldsymbol{\omega} - \nabla\left(\frac{P}{\rho} + \frac{1}{2}\mathbf{U}^2\right) + \nu\nabla^2\mathbf{U} + \mathbf{f}(\mathbf{x}, t), \tag{1}$$

together with the continuity equation

$$\nabla \cdot \mathbf{U}(\mathbf{x}, t) = 0. \tag{2}$$

Here $\boldsymbol{\omega} \equiv \nabla \times \mathbf{U}$ is the vorticity vector, P is the pressure, ρ is the fluid density and ν is the fluid (air) kinematic viscosity. Discretization of the domain is carried out by uniform division of the domain into N^3 grid points in all spatial directions. N is the number of grid points in each spatial direction. At these grid points, the turbulent fluid velocity is computed in the spectral space yielding the physical fluid velocity by an inverse Fourier transform. To achieve a stationary turbulence, the flow is driven by a forcing term $\mathbf{f}(\mathbf{x}, t)$ which is nonzero only for a few low wavenumber modes in the Fourier space. In this paper the average energy dissipation rate is assumed to be $\epsilon = 400\ \text{cm}^2/\text{s}^3$.

In the next step, the fluid velocity at the location of the kth droplet of radius $a^{(k)}$, denoted by $\mathbf{U}(\mathbf{Y}^{(k)}(t), t)$, is interpolated from the grid using the 6-point Lagrangian interpolation in each spatial direction, where $\mathbf{Y}^{(k)}(t)$ is the location of that droplet. The velocity of the kth droplet will be denoted by $\mathbf{V}^{(k)}(t)$. We assume the disturbance flows, resulted from the presence of the droplets suspended in the background turbulent flow, are localized in space, as the droplet size (10 to 60 μm in radius) is much smaller than the flow Kolmogorov scale (∼1 mm). If Stokes disturbance flows of all N_p droplets in the system are superposed appropriately [39] to satisfy approximately the no-slip boundary condition on the surface of each droplet, then the disturbance velocity felt by a given particle $\mathbf{u}^{(k)}$ can be solved from the following linear system

$$\mathbf{u}^{(k)} = \underbrace{\sum_{m=1}^{N_p}}_{m\neq k} \mathbf{u}_S\left(\mathbf{Y}^{(k)}(t) - \mathbf{Y}^{(m)}(t); a^{(m)}, \mathbf{V}^{(m)} - \mathbf{U}(\mathbf{Y}^{(m)}, t) - \mathbf{u}^{(m)}\right), \quad k = 1, 2, \ldots, N_p; \tag{3}$$

where the Stokes disturbance flow induced by the kth droplet is

$$\mathbf{u}_S(\mathbf{r}^{(k)}; a^{(k)}, \mathbf{V_p}^{(k)}) = \left[\frac{3}{4}\frac{a^{(k)}}{r^{(k)}} - \frac{3}{4}\left(\frac{a^{(k)}}{r^{(k)}}\right)^3\right]\frac{\mathbf{r}^{(k)}}{(r^{(k)})^2}(\mathbf{V_p}^{(k)} \cdot \mathbf{r}^{(k)}) + \left[\frac{3}{4}\frac{a^{(k)}}{r^{(k)}} + \frac{1}{4}\left(\frac{a^{(k)}}{r^{(k)}}\right)^3\right]\mathbf{V_p}^{(k)}. \tag{4}$$

Therefore, the disturbance velocity felt by a given droplet depends on the background turbulent flow, and positions and velocities of all other droplets in the system. It is important to notice that periodicity of the domain holds for particle positions and velocities, disturbance velocities $\mathbf{u}^{(k)}$, as well as for the flow.

Given $a^{(k)}$, $\mathbf{V}^{(k)}$, $\mathbf{Y}^{(k)}$ and $\mathbf{U}(\mathbf{Y}^{(k)}, t)$ at a specified time t, Eq. (3) represents a coupled linear system for $3N_p$ unknowns for the three components of the disturbance velocities ($\mathbf{u}^{(k)}$) of the N_p droplets. This droplet–droplet hydrodynamic interactions then yield an N-body problem.

As far as collision statistics are concerned, Ayala et al. [40] showed that one could truncate the summation in (3) and restrict the radius of influence of the disturbance flow caused by a mth droplet to within a distance $H_{trunc} \times a^{(m)}$. Their numerical experiments showed that the computed collision efficiency was insensitive to H_{trunc} if $H_{trunc} > 35$. In this paper, the dimensionless truncation radius is set to $H_{trunc} = 50$, which is more conservative than $H_{trunc} = 35$.

It is important to point out that this section of the code was the most computationally demanding step in the HDNS approach, taking up 80% of the total computational time (see [40]). Recently we developed a more efficient scheme to solve this system of equations based on an GMRes algorithm with preprocessing of coefficients [42], which we will discuss briefly in Section 3.

Once the undisturbed fluid velocity $\mathbf{U}(\mathbf{Y}^{(k)}(t), t)$ and the disturbance velocities $\mathbf{u}^{(k)}$/are computed at locations of all droplets, the droplets are advanced by solving their equation of motion which for the kth droplet becomes

$$\frac{d\mathbf{V}^{(k)}(t)}{dt} = -\frac{\mathbf{V}^{(k)}(t) - \left(\mathbf{U}(\mathbf{Y}^{(k)}(t), t) + \mathbf{u}^{(k)}\right)}{\tau_p^{(k)}} + \mathbf{g} \tag{5}$$

$$\frac{d\mathbf{Y}^{(k)}(t)}{dt} = \mathbf{V}^{(k)}(t) \tag{6}$$

where $\tau_p^{(k)} = \frac{2\rho_p(a^{(k)})^2}{9\mu}$ is the Stokes inertial response time of the kth droplet and $\mathbf{g}$ is the gravitational acceleration.

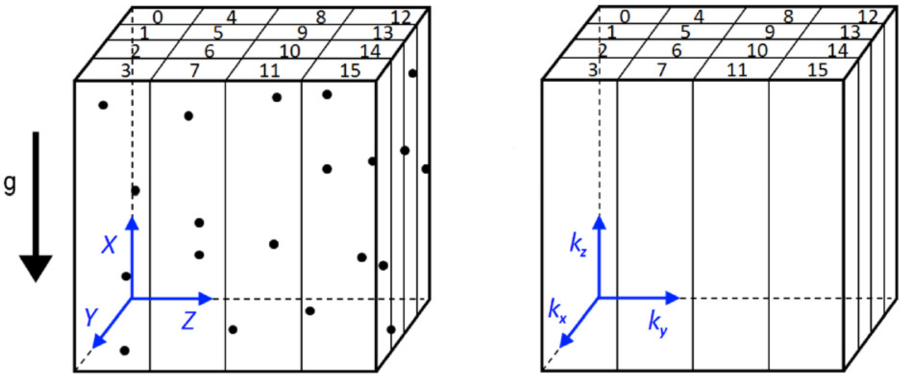


Fig. 1. A sketch showing 16 subdomains in 2D dd. Left panel shows fluid nodes and droplets in the physical space and each subdomain is assigned to an individual processor. X, Y and Z are the three directions of the coordinate system and $\mathbf{g}$ points in the gravity direction. Right panel shows how the domain is represented in the spectral space, noting the order of the wave number coordinate system due to the transposes for FFT calculations.

The algorithm for the HDNS approach can now be summarized in terms of the following 7 steps:

1. Advance the undisturbed fluid turbulence field $\mathbf{U}(\mathbf{x}, t)$ using a pseudo-spectral method;
2. Interpolate the undisturbed fluid velocities at the locations of the droplets, $\mathbf{U}(\mathbf{Y}^{(k)}(t), t)$ using a 6-point Lagrange interpolation in each direction;
3. Update the list of neighboring cells for each pair detection cell, in order to quickly detect all droplet pairs using the method of cell-index and linked list [43];
4. Solve the disturbance velocities $\mathbf{u}^{(k)}$ at the droplet locations using the GMRes method;
5. Advance the droplet velocities using the fourth-order Adams–Moulton method, and droplet locations using the fourth-order Adams–Bashforth method;
6. Detect droplet–droplet collision events and calculate other kinematic pair statistics.
7. Impose periodicity to the particle trajectory in the periodic computational domain and the non-overlapping condition to the particles (necessary condition for the system (3), see Ayala et al. [40]).

In the next section we will discuss the parallel implementation for each task and develop a complexity analysis.

3. Parallel implementation using 2D dd and a complexity analysis

Our MPI implementation is based on a 2D dd previously adopted for efficient parallel implementation of Fast Fourier Transform (FFT) in the pseudo-spectral simulation of fluid turbulence by Ayala and Wang [44]. In their work, they argued that the 3D dd performance tends to saturate and deteriorate with excessive communication, leaving 2D dd to be the best option to parallelize the FFT for large number of processors since 1D dd is limited to few processors (number of processors has to be less than the number of grid points along the decomposed domain).

Here we extend the 2D dd to the dispersed phase. Since the dispersed system is homogeneous on average (but instantaneously the particle distribution is not uniform in space) and the dilute droplet concentration implies one-way coupling between the dispersed phase and the background air flow, we anticipate, on average, a reasonable computational load balance using a static domain decomposition parallelization scheme.

Fig. 1 (left panel) shows the computational domain in the physical space that is decomposed in the two directions perpendicular to gravity in order to minimize the number of droplets crossing the subdomain boundaries. Consequently this will likely minimize the data communication leading to a higher speed-up. MPI is used to communicate data between the subdomains whenever the solution or data collection procedure requires data from non-local processors. In the right panel of Fig. 1 we show the domain in the spectral space after performing 3D FFT. Note the order of the wave number coordinate system due to the transposes for FFT calculations. The spectral space is only used to solve the flow.

In the remainder of this section, we will describe in detail the parallel implementation issues for each task as described in Section 2. A complexity analysis is also developed to better understand scalability performance and to forecast the parallel performance of our code for any problem size. The parameters determining the problem size include flow grid resolution N, the number of particles N_p, and particle sizes r_1 and r_2, and the number of processors P. The execution wall-clock time can be decomposed into computations (*i.e.*, the number of floating point operations), data copy operations, and communications.

It is now necessary to introduce several elemental times: $t^{\text{comp}}_{(N_{\text{size}})}$ is the computation time per floating point operation (FLOP), $t^{\text{copy}}_{(N_{\text{size}})}$ is the memory-to-memory copy time per word, $t^{\text{comm}}_{(P)}$ is the time for transmission of a single word from one processor to another, and $t^{\text{startup}}_{(P)}$ is the startup or latency time during communication. A complete communication between two processors involves two steps, a send and a receive. These communication elemental times refer to each of those steps.

A subtle complexity is that the elemental times per data depend on the data size. Therefore, in contrast to Ayala and Wang [44] who assumed the elemental times to be constant, in this paper we allowed the elemental times to depend on how large the problem at hand is. Thus, $t^{\text{comp}}_{(N_{\text{size}})}$ and $t^{\text{copy}}_{(N_{\text{size}})}$ depend on the array size (N_{size}) that they are operating on; while $t^{\text{comm}}_{(P)}$ and $t^{\text{startup}}_{(P)}$ depend on the number of processors involved in the simultaneous communication. These elemental times due to communications depend on how busy the network of the machine is. When obtaining them, the communication bandwidth could be shared with other users, so the times will be representative of what a user can actually achieve. If there are few total FLOP in a code line (or copying few words from one array to

Table 1
Estimated elemental times on Stampede (or Yellowstone) for the complexity analysis.

$t^{\text{comp}}_{(N_{\text{size}})}$	$0.04N^{0.20}_{\text{size}}$ ns/FLOP 0.21 for FFT
$t^{\text{copy}}_{(N_{\text{size}})}$	$0.37N^{0.20}_{\text{size}}$ ns/word
$t^{\text{comm}}_{(P)}$	$0.98P^{0.40}$ ns/word
$t^{\text{startup}}_{(P)}$	$0.83P^{0.27}$ μs
$t^{\text{comp}}_{\text{Retrieval}}$	10 ns
$t^{\text{copy}}_{\text{Retrieval}}$	40 ns
$t^{\text{reduce}}_{(C_{\text{size}},P)}$	$C_{\text{size}} \log_2(P) t^{\text{startup}}_{(P)}$ μs

another), the computation time (or memory-to-memory copy time) is constant due to data retrieval from and to machine memory (see Appendices). In this case, the computation time and memory-to-memory copy time are defined as $t^{\text{comp}}_{\text{Retrieval}}$ and $t^{\text{copy}}_{\text{Retrieval}}$.

In our code, there are a number of MPI_ALLREDUCE commands. This command combines data from all processes and distributes the result back to all processes. Its elemental time should depend on the number of processors P and the data size communicated by MPI_ALLREDUCE. In this code, the data size is small (it varies from 2 to 680 words), thus the startup time ($t^{\text{startup}}_{(P)}$) will dominate. This elemental time can be estimated by $t^{\text{reduce}}_{(C_{\text{size}},P)} = C_{\text{size}} \log_2(P) t^{\text{startup}}_{(P)}$, where C_{size} depends on the data size. When the data size is about 2 words, C_{size} is approximately 0.5. We would like to point out that the cost of MPI_ALLREDUCE commands with small messages is negligible as long as the number of processors involved is small but when it is large, the cost is not trivial and increases rapidly with P.

In Appendix we discuss how to estimate the elemental times shown in Table 1. We must point out that this was not a trivial task and what we propose is just one approach that work reasonably well here. The elemental times were obtained using the supercomputer Stampede at Texas Advanced Computing Center, a Dell PowerEdge C8220 Cluster with Intel Xeon Phi coprocessors. We also tested on the supercomputer Yellowstone at NCAR-Wyoming Supercomputing Center. Since their CPU hardware configurations are very similar, we found that the elemental times are also similar for the two machines. It is important to mention that the elemental times have their complexity as they depend on many factors such as machine, network topology and contention, affinity, compiler, computer programming language, array rank, array format (row-major or column-major formats, or also known as C order and Fortran order), vector stride, number of FLOP in a code line, number of processors, to name a few. The experiments performed in Appendices to estimate them have their own characteristics in terms of array ranks and array formats which do not match the characteristics of other subroutines in our code. To achieve simplicity of the expressions, we only empirically adjust N_{size} to properly consider the differences.

We shall reiterate and stress that there are many factors that affect the overall code execution time, including machine, network topology and contention, affinity, compiler, environmental variables, computer programming language, coding style, array rank, array format, vector stride, number of FLOP in a code line, number of processors, cache memory size, and MPI setting. However, the way we developed the complexity analysis is such that those factors could be incorporated, approximately and empirically, by re-measuring the elemental times for a specific set of configurations considered. We caution that this task might not be trivial as some factors may not be independent. In Appendix A we outlined a procedure to obtain them, which appears to work reasonably well for the supercomputers we used.

3.1. Background turbulent flow

The background turbulent flow was solved using a pseudo-spectral method. Details of the method can be found in Wang and Maxey [45], Wang et al. [46], and Zhou et al. [47]. The scheme advances the flow in the spectral space, while the non-linear term in the Navier–Stokes equation is computed in the physical space. For the complexity analysis, the method can be decomposed into the following steps:

1. Perform a backward FFT to convert the velocity in the spectral space ($\hat{\mathbf{U}}(\mathbf{k}, t)$) to the physical space ($\mathbf{U}(\mathbf{x}, t)$);
2. Compute vorticity in the spectral space $\hat{\boldsymbol{\omega}}(\mathbf{k}, t) = i\mathbf{k} \times \hat{\mathbf{U}}(\mathbf{k}, t)$;
3. Perform a backward FFT to convert the vorticity in the spectral space ($\hat{\boldsymbol{\omega}}(\mathbf{k}, t)$) to the physical space ($\boldsymbol{\omega}(\mathbf{x}, t)$);
4. Compute the non-linear term in the physical space $\mathbf{N_1}(\mathbf{x}, t) = \mathbf{U}(\mathbf{x}, t) \times \boldsymbol{\omega}(\mathbf{x}, t)$;
5. Perform a forward FFT to convert the non-linear term in the physical space ($\mathbf{N_1}(\mathbf{x}, t)$) to the spectral space ($\hat{\mathbf{N_1}}(\mathbf{k}, t)$);
6. Advance the flow in time using Adams–Bashforth scheme for the non-linear term and Crank–Nicholson scheme for the viscous term;
7. Add energy to the flow through the forcing term $\mathbf{f}(\mathbf{x}, t)$ using the Euler scheme;
8. Enforce conjugate symmetry in the spectral space for the $k_x = 0$ plane (this symmetry should be automatically satisfied when taking a real to complex transform but we enforce it to eliminate any round off error);
9. Reduce/Eliminate aliasing errors by truncating a portion of the large-wavenumber Fourier modes.

The most time-consuming parts are the 9 FFTs performed in steps 1, 3, and 5. It takes about 90% of the computational time [8]. For the FFT we adopted the efficient parallel implementation as developed by Ayala and Wang [44]. This subroutine requires unavoidable global communications among processors due to the inherent non-locality of the Fourier Transform. Calculations in steps 2, 4, and 6 are completely local and they are performed in parallel by the processors.

Other steps now deserve some attention. In Step 7, the forcing term only involves a few low wavenumbers that are stored in specific processors (in right panel of Fig. 1, only in processors 0 and 12. Processor 0 handles the low wave numbers with positive values of k_y and processor 12 handles the low wave numbers with negative values of k_y). No communications are involved but as this is a localized computation, we chose not to parallelize it. We decided to allow only two processors to handle the computation for the forcing terms, imposing a constraint on the number of processors along each of the decomposed coordinate systems (Y and Z); namely, $P_y < N/6$ and $P_z < N/3$, where P_y and P_z are the number of processors along the Y and Z axis, respectively, and N is the number of fluid nodes along each of the coordinates axes. This amounts to $P_{\text{force}} = N^2/18$, the maximum number of processors restricted by the energy forcing subroutine. It is important to point out that this restriction could be relaxed with simple modifications to the subroutine. However, as we will show later, this maximum number of processors (P_{force}) is usually sufficient considering the possibility that the FFT calculation can saturate and deteriorate when a larger number of processors are used.

Concerning Step 8, the symmetry plane $k_x = 0$ is shared by a subset of processors (in right panel of Fig. 1, processors 0, 4, 8 and 12), therefore it involves communication within certain processors (while others are left idle) on only a small volume of the data. As far as the dealiasing step, wavenumber modes that satisfies $|\mathbf{k}| \geq (N/2 - 1.5)$ are set to zero to control the aliasing errors [45]. In terms of parallel execution, this implies an unavoidable load imbalance.

Taken together, the execution time for the flow simulation step can be expressed using the following complexity analysis,

$$t_{\text{FLOW}} = 9\,t_{\text{FFT}} + \underbrace{\left(72\,t^{\text{comp}}_{(N^3/P)} + 22\,t^{\text{copy}}_{(N^3/P)}\right)\frac{N^3}{P}}_{\text{Advance Flow \& Dealiasing}} \tag{7a}$$

$$+ \underbrace{14461\,t^{\text{comp}}_{(450)} + 1034\,t^{\text{copy}}_{(450)} + 540\,t^{\text{comp}}_{(N^3/P)} + 20\,t^{\text{copy}}_{(N^3/P)}}_{\text{Forcing}} \tag{7b}$$

$$+ \underbrace{3\left[\left(6\frac{N^2}{\sqrt{P}} + 10N\right)t^{\text{comp}}_{(N^2/\sqrt{P})} + \left(4\frac{N^2}{\sqrt{P}} + 2N\right)t^{\text{copy}}_{(N^2/\sqrt{P})} + 4\left(\frac{N^2}{\sqrt{P}} + N\right)t^{\text{comm}}_{(\sqrt{P})} + 4\,t^{\text{startup}}_{(\sqrt{P})}\right]}_{\text{Symmetry (noise elimination)}}, \tag{7c}$$

where t_{FFT} was derived in Ayala and Wang [44] as

$$t_{\text{FFT}} = \frac{5}{2}\frac{N^3\log_2 N^3}{P}t^{\text{comp}}_{\text{FFT}} + 6\frac{N^3}{P}t^{\text{copy}}_{(N^3/P)} + 4\frac{N^3}{P}t^{\text{comm}}_{(P)} + 4\left(\sqrt{P} - 1\right)t^{\text{startup}}_{(P)}, \tag{8}$$

where P is the number of processors, and N is the grid resolution. For the complexity analysis expressions in this paper, it was assumed that $P_y = P_z$. The above complexity analysis shows that t_{FFT} typically takes 70% to 95% of the wall-clock of the flow simulation, before the scalability deteriorates at large P. Chen and Shan [8] found it to be 94% for their implementation on a Connection Machine with minimal communication. The second term in (7a) represents the computation time for advancing the flow and dealiasing. The operations to advance flow are local and performed at each grid point. The dealiasing operation is done only in a portion of the domain, thus certain processors perform the operation while others are idling. The terms in (7b) are due to the forcing scheme. This is an unparallelizable portion (it does not depend on P) but it represents about less than 1% of the total computational load. The forcing preparation involves operations with array sizes of 450 elements. Later, when the forcing is applied on the flow field, it is necessary to operate on the local flow array of size N^3/P. Therefore, the two different problem sizes appear in the elemental times. Finally, the terms in (7c) are due to enforcing symmetry, with the local data size being $N^2/\sqrt{P}$. Here, communication is necessary between the two neighboring processors along one decomposed direction (k_y direction in left panel of Fig. 1). The transmission time and the start-up time depend on $\sqrt{P}$ since the communication is done over processors handling this plane only.

3.2. Interpolation

Quite a few different interpolation techniques have been developed in the past, see a brief review of the commonly employed methods in [41]. There has been some work on the accuracy of the interpolation methods for particle tracking in turbulence (see for instance, [48, 49]). Everyone is aware that the best interpolation methods in terms of accuracy are high-order methods but they are computationally expensive. Therefore, it is important to make the best compromise between accuracy and computational cost. To the best of our knowledge, this has not been done rigorously. The tendency is to look at accuracy quantitatively while observing computational cost qualitatively. With the advance of supercomputer capability, there is no final answer on the best parallel interpolation method yet.

In our code we use the high-order scheme six-point Lagrangian interpolation in each spatial direction. Interpolation of undisturbed flow velocity to particle centers is a task that necessitates communication. The 6 interpolation points require communication of grid points near subdomain boundaries (3 grid layers from right and top and 2 grid layers from left and bottom).

As illustrated in Fig. 2, data communication near the subdomain boundaries could be handled in two different manners. The left panel shows a sequential (or successive) communication scheme where the grid velocity data of dimensions $N \times 2 \times N/P_z$ and $N \times 3 \times N/P_z$ (for each velocity component) are first communicated to the right (the positive y direction) and left (the negative y direction), respectively. At this point, each processor contains velocity data of the size $N \times (N/P_y + 5) \times (N/P_z)$. In the second communication step, the grid velocity data of dimensions $N \times (N/P_y + 5) \times 2$ are sent across the top subdomain boundary (the positive z direction) and the velocity data of dimensions $N \times (N/P_y + 5) \times 3$ are sent across the bottom boundary (the negative z direction). These extended data sizes take care of the corner regions automatically. In the end, the size of each velocity data within a processor is $N \times (N/P_y + 5) \times (N/P_z + 5)$. This algorithm requires only four communication pairs (i.e., two in each communication step). Since it is sequential, this scheme requires a blocking after the right-left communication step.

Alternatively, in the right panel of Fig. 2, we show a non-sequential communication scheme where data to and from the 8 different neighboring subdomains are communicated independently. In this approach all the communications could be performed simultaneously without blocking, but eight communication pairs are needed.

Our numerical experiments show that for the range of parameters and decomposition sizes (i.e., P_y and P_z) we have considered, there is only $\sim$1% difference in the overall run-time between the two schemes. This shows that the communication overhead of the non-sequential method compensates the blocking and extra copy time required for the sequential scheme. In the rest of this paper, we will make use of the non-sequential communication scheme (right panel of Fig. 2) as we found its performance to be less machine dependent.

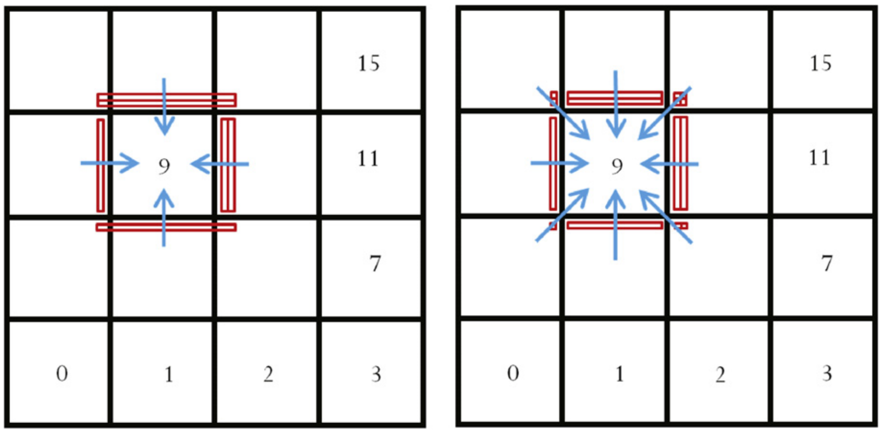


Fig. 2. Sketches to show two different communication strategies for fluid velocity interpolation. Red lines show the layers of grid velocity data to be communicated. Left panel: sequential communication involving 4 communication pairs. Right panel: non-sequential communication involving 8 communication pairs.

Examination of the velocity interpolation code led to the following complexity analysis,

$$t_{\mathrm{INTERP}} = \underbrace{10\left(t^{\mathrm{copy}}_{(N^3/P)} + 2\,t^{\mathrm{comm}}_{(P)}\right)\frac{N^2}{\sqrt{P}} + 16\,t^{\mathrm{startup}}_{(P)} + t^{\mathrm{copy}}_{(N^3/P)}\frac{N^3}{P}}_{\text{Communication halo regions}} \tag{9a}$$

$$+\underbrace{864\,t^{\mathrm{comp}}_{(N^3/P)}\frac{N_p}{P}}_{\text{Computation}}. \tag{9b}$$

The first term in (9a) represents the transmission of data across the subdomain boundaries, while the second term represents the latency related to the communications from the 4 sides and the 4 corners of the subdomain. All processors are busy while the communication takes place, therefore t^{comm} and t^{startup} depend on P. The third term in (9a) accounts for the copy of N^3/P data to form the extended array. Relatively speaking, the data transmission for the corners and the copy of communicated data to form the extended array are negligible and they are not included in the equation. The term (9b) represents the actual interpolation calculations, with t^{comp} dealing with data of size N^3/P.

We would like to mention in passing that lower order interpolation schemes could be used to reduce the data communication requirements (i.e., size of the halo regions) and, therefore, its execution time. However, the accuracy of the particle tracking could be at risk. Other interpolation schemes will be tested in the future to rigorously reach an optimal balance between parallel performance and accuracy.

3.3. The cell-index and linked list algorithm

In order to efficiently locate a droplet and its neighboring droplets within the truncation sphere of a given droplet, we make use of the cell-index method and the concept of linked lists [43]. This is implemented by dividing the domain into small cells (N^3_{cell} in total) and keeping track of droplet indices inside each cell. This speeds up the process of finding neighbor droplets, without affecting the simulation results. This subroutine is used to search all particle pairs for local hydrodynamic interactions, collision detection, overlapping detection, or for computing pair statistics. The search not only involves particle pairs residing in the host processor but also pairs with one particle in the host processor and a second particle in neighboring cells. The regions where the second particle could be located are called "halo regions" and they are represented by the gray area in Fig. 3. The thickness of this halo region is related to the largest truncation radius in the system, $H_{\mathrm{trunc}} \times a_{\max} = 50a_{\max}$, where $a_{\max}$ is the maximum droplet radius.

A bookkeeping provides the following complexity analysis for this section of the code,

$$t_{\mathrm{HEAD\text{-}LIST}} = \underbrace{\left(\frac{N^3_{cell}}{P}\right)t^{\mathrm{copy}}_{(N^3_{cell}/P)}}_{\text{Assigning ZEROS}} + \underbrace{\left(70\,t^{\mathrm{comp}}_{(N_p/2P)} + 18\,t^{\mathrm{copy}}_{(N_p/2P)}\right)\frac{N_p}{P}}_{\text{Detecting particles}} \tag{10a}$$

$$+\underbrace{\left(16\,t^{\mathrm{comp}}_{(N_p/2P)} + 20\,t^{\mathrm{copy}}_{(N_p/2P)} + 144\,t^{\mathrm{comm}}_{(P)}\right)\left(50a_{\max}\frac{L}{\sqrt{P}} + 2500a^2_{\max}\right)\frac{N_p}{L^2} + 32\,t^{\mathrm{startup}}_{(P)}}_{\text{Communication halo regions}}. \tag{10b}$$

The first term of (10a) accounts for the need to initialize a zero value to the large three dimensional array HEAD and other vectors. Here we highlight the importance of controlling the number of copies in a code. Although this first term takes only a single line in a Fortran 90 code, it represents a non-negligible operation. The second term measures the cost of computations needed to detect particles in the halo regions (before communication) and within a cell (after communication). Here, the data size for t^{comp} and t^{copy} is N_p/P. However, there are some operations involving other lower rank arrays and arrays of smaller sizes, therefore an empirical 0.5 factor is introduced. The third term captures the communication of the halo regions, where the transmitted particle data size is proportional to the halo-region area on the four sides ($50a_{\max}L/\sqrt{P}$) and the halo-region area for the corners ($2500a^2_{\max}$). There are 16 communication pairs involved, twice for

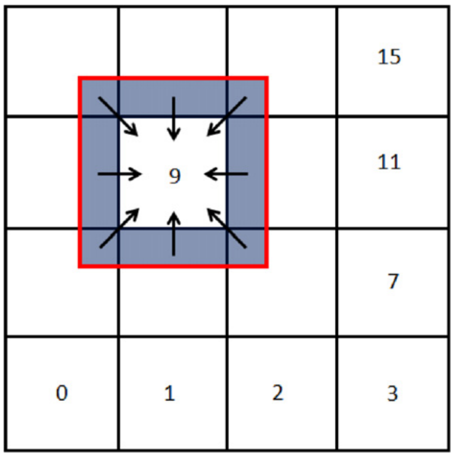


Fig. 3. Shaded regions around a subdomain 9 denote all halo regions in 2D domain decomposition. All particle data in these regions are made available for the subdomain 9 via communications from 8 neighboring subdomains. Arrows indicate the direction of data communication.

each of the 4 sides and 4 corners. It is important to mention that, in this subroutine, the communicated data for particles residing in the halo regions include not only current particle location (needed for the cell-index array, HEAD; and linked list array, LIST) but also current particle velocity, particle velocity and position in several previous time steps, and a droplet global index. The global index is a unique indicator of a droplet which will be passed onto a new host subdomain if the droplet changes its subdomain. This enables us to track a specific particle over the whole domain, if needed.

3.4. *Disturbance flow velocities*

For each given droplet of radius a, interactions with all neighboring droplets located within a distance of $H_{\text{trunc}} \times a$ must be considered. For convenience, the width of the detection cells is designed such that the truncation sphere can at most cover a space not larger than the one belonging to the immediate neighboring subdomains. This assumption amounts to setting an upper-bound on the number of subdomains (therefore, processors) in each decomposed direction, as

$$P_y^{\text{HI-detect}} = P_z^{\text{HI-detect}} = \frac{L}{H_{\text{trunc}} \times a_{\max}} \tag{11}$$

where $P_y^{\text{HI-detect}}$ and $P_z^{\text{HI-detect}}$ are the maximum number of processors permissible along the decomposed y and z direction, respectively; and L is the computational domain size. With this constraint, the total number of processors should not exceed $P_{\text{HI-detect}} = (L/(H_{\text{trunc}} \times a_{\max}))^2$. Since, the droplet radius is in the range of 10 to 60 μm, this requirement does not pose a severe restriction on the number of cores and has the benefit of limiting the data communication to 8 nearest neighbor subdomains.

We have recently developed a more efficient scheme to solve (3) based on the GMRes algorithm with preprocessing of coefficients [42]. This system of equations could be represented by $\mathbf{Ax} = \mathbf{b}$. The first phase of the subroutine is to compute the coefficients of the sparse matrix $\mathbf{A}$ and the vector $\mathbf{b}$ and to initialize the GMRes method. Once the initialization and the preprocessing of coefficients are completed, the GMRes iterative process begins. Torres et al. [42] found that the number of iterations depends on the problem setting, however they showed that it depends only weakly on N_p and droplet sizes (a_1, a_2). A typical iteration number is $\sim$10 for a convergence to 10^{-6} relative error. While it is possible to use other iterative schemes to reduce the data communication cost, they may not converge as quickly [42].

Since this subroutine could be of less interest to the general readers, the full complexity analysis equations and the detailed explanation of their derivations are in Appendix B. In there we compare our findings against the recently found by Onishi et al. [26] on minimizing the cost for particle interaction calculations.

3.5. *Advancing particle equation of motion*

This step is completely local without data communication and is expected to scale linearly with P. Basically, we advance the particle velocity and location, and update the arrays containing particle velocity and position at previous steps required for the fourth-order Adams–Bashforth schemes. Similarly to the previous subroutine, an empirical factor of 180 is used for t^{copy} and t^{comp}. The complexity analysis for this step is

$$t_{\text{ADVANCE}} = \underbrace{\left(200\, t^{comp}_{(180N_p/P)} + 40\, t^{copy}_{(180N_p/P)}\right) \frac{N_p}{P}}_{\text{Local computations}}. \tag{12}$$

3.6. *Post-processing droplet statistics*

Post-processing of droplet data might also require data communication depending on what statistics are being processed. Local particle data are readily available to be printed out without communication between subdomains. On the other hand, radial quantities (or pair quantities) will require communication and book-keeping to gather the data on a designated core(s) for print-out. Also, space-averaged statistics typically require a local summation followed by a global data reduction on a designated core(s). If output is needed for a large quantities of data ("large" will depend on communication bandwidth, local storage space, etc.), it would be convenient if we designate several cores instead of a single core. This will alleviate the unpredictable and hard-to-handle desynchronization issues during data collection.

Specifically, we are interested in particle pair statistics which are typically more time consuming than single particle statistics. Since this subroutine could be of less interest to the general readers, the full complexity analysis equations and the detailed explanation of their derivations are in Appendix C.

3.7. Particle domain relocation (periodicity) and non-overlapping conditions

In this subroutine we handle all particle migrations between subdomains and the non-overlapping condition to the particles. Since periodicity to particle trajectory in the 3D periodic computational domain implies also migration between one subdomain to another, this subroutine also handles it. The non-overlapping condition is a necessary condition for the system (3), see Ayala et al. [40].

Particle motion between subdomains requires a transfer of migrated particle's data from a source subdomain to host subdomain. The droplet data communication is implemented by proper use of global droplet indices and careful rearrangement of droplet data when droplets are leaving or entering a subdomain. At each time step, after advancing the particle location, migrating particles are detected and their data are stored in 8 different buffers (4 sides and 4 corners) depending on where their new location is. Then the buffered lists are communicated between source and host subdomains. Departed particles are removed from local particle list and arrived particles are added. In this step we take care of periodicity of the particles, so that particles which are moving the domain on one side of the box are brought into their new host domain on the other side of the computational box.

For the non-overlapping condition, the HEAD and LIST arrays have to be updated due to the new particle locations. Therefore, inside the overlapping subroutine, the Head-List subroutine is called. After detecting all overlaps in the whole domain, an MPI_ALLREDUCE operation is needed to count the total number of overlappings. If an overlapping is detected, the pair is removed and randomly relocated in the domain. Then, the non-overlapping condition is checked again until no overlapping is detected.

The complexity analysis for these two components is

$$t_{\text{PERIOD-OVERL}} = \underbrace{t_{\text{HEAD-LIST}}}_{\text{Update}} + \underbrace{t^{\text{reduce}}_{(0.5,P)}}_{\text{Collecting overlaps}} + \underbrace{32\, t^{\text{startup}}_{(P)}}_{\text{Communicate halo regions}} + \underbrace{3^3 \left(25\, t^{\text{comp}}_{(N_p/2P)} + t^{\text{copy}}_{(N_p/2P)}\right) \frac{N_p}{P}}_{\text{Needed operations}} . \qquad (13)$$

The first and second terms delineate the time for enforcing the non-overlapping condition. Since the particle volume fraction is small, the number of particle overlappings per time step is small. Consequently, the number of FLOP and copies needed for actual overlapping detection is negligible, thus $t_{\text{HEAD-LIST}}$ and the MPI_ALLREDUCE operation dominate the time. The third and fourth terms of the equation accounts for the time of the periodicity subroutine. As we purposely aligned the gravity direction along the undecomposed direction (x) of the domain, the number of droplets moving between subdomains is reduced. That along with the small particle volume fraction makes data transmission to be negligible. In this subroutine, two steps of the communication are needed, leading to 16 communications (due to the 8 different halo regions, see Fig. 3). In the first, only the number of crossing particles is transmitted; while in the second, we then move the crossing particle data into the corresponding new host subdomain.

4. Performance analysis

4.1. Scalability

The resulting MPI code was run on Stampede at Texas Advanced Computing Center (TACC), a Dell Linux Cluster based on 6400+Dell PowerEdge server nodes with 16 cores per node (http://www.tacc.utexas.edu/resources/hpc/stampede). The core frequency is 2.7 GHz and supports 8 floating-point operations per clock period with a peak performance of 21.6 GFLOPS/core. Each node has an accelerator Intel Xeon Phi Coprocessor (MIC Architecture), which was not used for performance testing since our code is designed with MPI commands only. Nodes are interconnected with Mellanox FDR InfiniBand technology in a 2-level (cores and leafs) fat-tree topology. The 56 GB/s FDR InfiniBand interconnect consists of Mellanox switches, fiber cables and HCAs (Host Channel Adapters). The operating system of the machine is Linux, we used Intel Fortran compilers, and for the Message Passing Interface (MPI) libraries we use MVAPICH2.

In addition, and for comparison, we also run our code on a recently established supercomputer, Yellowstone, at NCAR-Wyoming Supercomputing Center (http://www2.cisl.ucar.edu/resources/yellowstone). In general, we found that the two computers yield a similar performance in terms of execution wall-clock time. The two machines have similar hardware.

We measured the actual wall-clock time for each subroutine described previously. We repeated the execution of each subroutine 50 times, and the averaged execution time will be presented. In Fig. 4 we show the percentages of wall-clock time spent on different subroutines for the whole HDNS approach. Two different cloud microphysics settings are considered. The pseudo-spectral simulation of the background turbulent flow takes the most amount of time, nearly 80%. In contrast, the time to advance the particle equation of motion is less than 1% and insignificant. In between, the other parts of the code take different percentages: roughly 5% to 10% for disturbance flow velocities and for the velocity interpolation, roughly 1% to 5% for post-processing and for periodicity and non-overlapping, and 1% or less for the cell-index and linked list. The relative percentages change with the number of processors and the problem size. The performance of the tasks for the particulate phase is rather satisfactory given that the domain decomposition is used to scale the code, as the particle decomposition would be more natural for scaling the particle dynamics. The flow simulation, disturbance flow velocities, and velocity interpolation are the top three demanding tasks, and they all involve significant data communications. Another important observation is that the postprocessing time percentage increases rather rapidly with the number of processors, due to the MPI_ALLREDUCE data-reduction commands needed to gather the statistics.

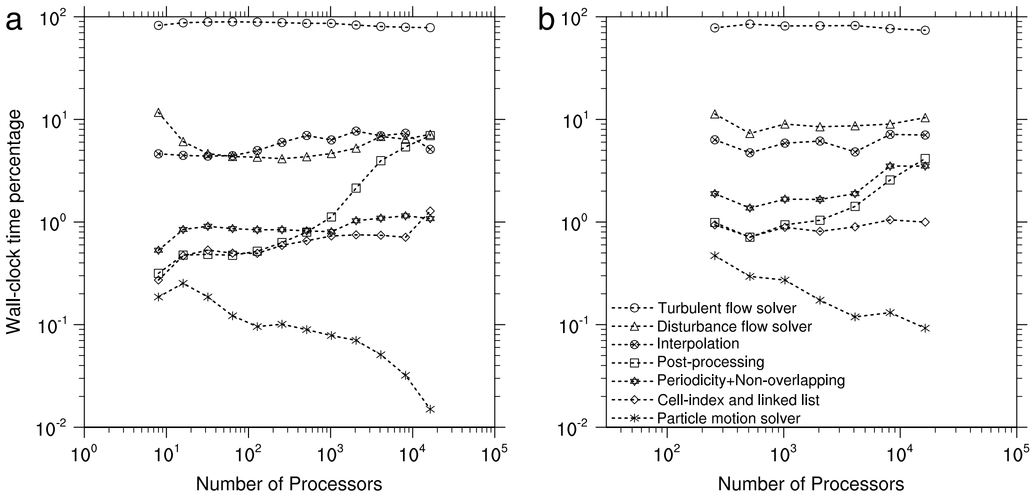


Fig. 4. Relative percentages of the wall-clock time spent on each subroutine for two cloud microphysics parameter settings: (a) 512^3, 45–50 μm, 730,000 particles (LWC ∼2.2 g/m^3), (b) 1024^3, 20–30 μm, 16,500,000 particles (LWC ∼ 1.0 g/m^3). Here LWC indicates the liquid water content.

In Figs. 5 and 6, we compare the prediction by the complexity analysis against the measured wall-clock times for different tasks. The data points for number of processors range 3 decades because we would like to show the capability of the complexity analysis to predict wall-clock times for a wide range of P. A line of slope −1 is plotted to serve as reference for ideal scalability. In all cases there is a satisfactory agreement between the complexity analysis and the measured time, in both the good scalability region and the saturation–deterioration region. The saturation–deterioration occurs for large P except for advancing the particle equation of motion which is communication-free. This saturation–deterioration originates from the increasing communication (due to latency) as P is increased, namely, communication does not scale as well when compared to computation. As it can be observed in Fig. 6, the flow and particle sections of the code saturates and deteriorates at different number of processors due to the different types of data communication. The flow calculations saturate and deteriorate when the 3D FFT subroutine does. Ayala and Wang [44] derived an expression for the maximum number of processors dividing the good scalability region and the saturation–deterioration region, as

$$P_{\mathrm{FFT}} = \left[\left(\frac{5}{4} \log_2 \left(N^3\right) t^{\mathrm{comp}}_{\mathrm{FFT}} + 3 t^{\mathrm{copy}}_{(N^3/P_{\mathrm{FFT}})} + 2 t^{\mathrm{comm}}_{(P_{\mathrm{FFT}})} \right) \frac{N^3}{t^{\mathrm{startup}}_{(P_{\mathrm{FFT}})}} \right]^{2/3}. \tag{14}$$

In their original expression, the elemental times were constant. In the current study, we found that the elemental times depend on array sizes and number of processors, in this case, they depend on P_{FFT}, making the expression implicit. Solving Eq. (14) for P_{FFT}, for the two grid resolutions considered in Fig. 6, we find that P_{FFT} is 8192 (using 2^n processors) for the 512^3 grid, while P_{FFT} for the right panel is 32768 (using 2^n processors) for the 1024^3 grid. Both are close to what are observed in Fig. 6.

We now derive the maximum number of processors for the disturbance flow velocities before a severe saturation–deterioration occurs, as it is the most time consuming task for the particulate phase. For simplicity, we assume the elemental times to be constant and follow similar procedure as in Ayala and Wang [44] in setting the derivative of the theoretical execution time to zero to obtain the minimum execution time before our strategy starts to saturate and deteriorate. We also simplified terms that are small in the cloud microphysics context. The resulting expression for the maximum number of processors

$$P_{\mathrm{HI}} = \left[10.42 \frac{a_{\max}}{L} \frac{N_p}{(3+n)} \frac{\left(28 t^{\mathrm{copy}}_{(180N_p/P_{\mathrm{HI}})} + 6 t^{\mathrm{comm}}_{(P_{\mathrm{HI}})}\right)}{t^{\mathrm{startup}}_{(P_{\mathrm{HI}})}} + 1.66 \left(\frac{N_p}{(3+n)} \frac{\left((20+9n) t^{\mathrm{comp}}_{(180N_p/P_{\mathrm{HI}})} + 18 t^{\mathrm{copy}}_{(180N_p/P_{\mathrm{HI}})}\right)}{t^{\mathrm{startup}}_{(P_{\mathrm{HI}})}} \right)^{1/2} \right]^2. \tag{15}$$

Solving the implicit Eq. (15) for P_{HI}, we find that for the case of the left panel in Fig. 6, P_{HI} is 4096 (using 2^n processors), while for the right panel P_{HI} is 65,536 (using 2^n processors). Once again, both are close to what are observed. It is important to point out that the saturation–deterioration of the particle section could happen at smaller or larger number of processors than the flow section depending on the problem setup. Nevertheless, as long as the turbulent flow simulation dominates the execution time, the overall saturation–deterioration of the code will be associated to the saturation–deterioration of the flow calculations. However, as we will show later, there are special cases at which the disturbance flow velocities calculations dominate. We would like to stress that the most efficient use of the code occurs before any saturation–deterioration occurs, thus Eqs. (14) and (15) should be used together to estimate the maximum number of processors to use.

We also tested our code on a different supercomputer, Kraken (http://www.nics.tennessee.edu/computing-resources/kraken/) from the National Institute for Computational Sciences (NICS) at the University of Tennessee. Fig. 7 shows the results. There is a good scalability of our code for this machine as well. For comparison against the scalability on Stampede, the complexity analysis for Stampede is included. Due to the different machine architectures, our code is slower and saturates (and deteriorates) earlier when running on Kraken. Our complexity analysis should also work on Kraken if the elemental times associated to this machine are obtained. Another interesting point to mention is that the performance on Stampede of our subroutine for particle calculations, before it saturates and deteriorates, tends to scale better than the ideal P^{-1} scaling. The processors on this machine make operations (computation or copy) faster as the array size within a processor becomes smaller which did not happen on Kraken. Eq. (B.2) also shows this better scaling. By examining the terms

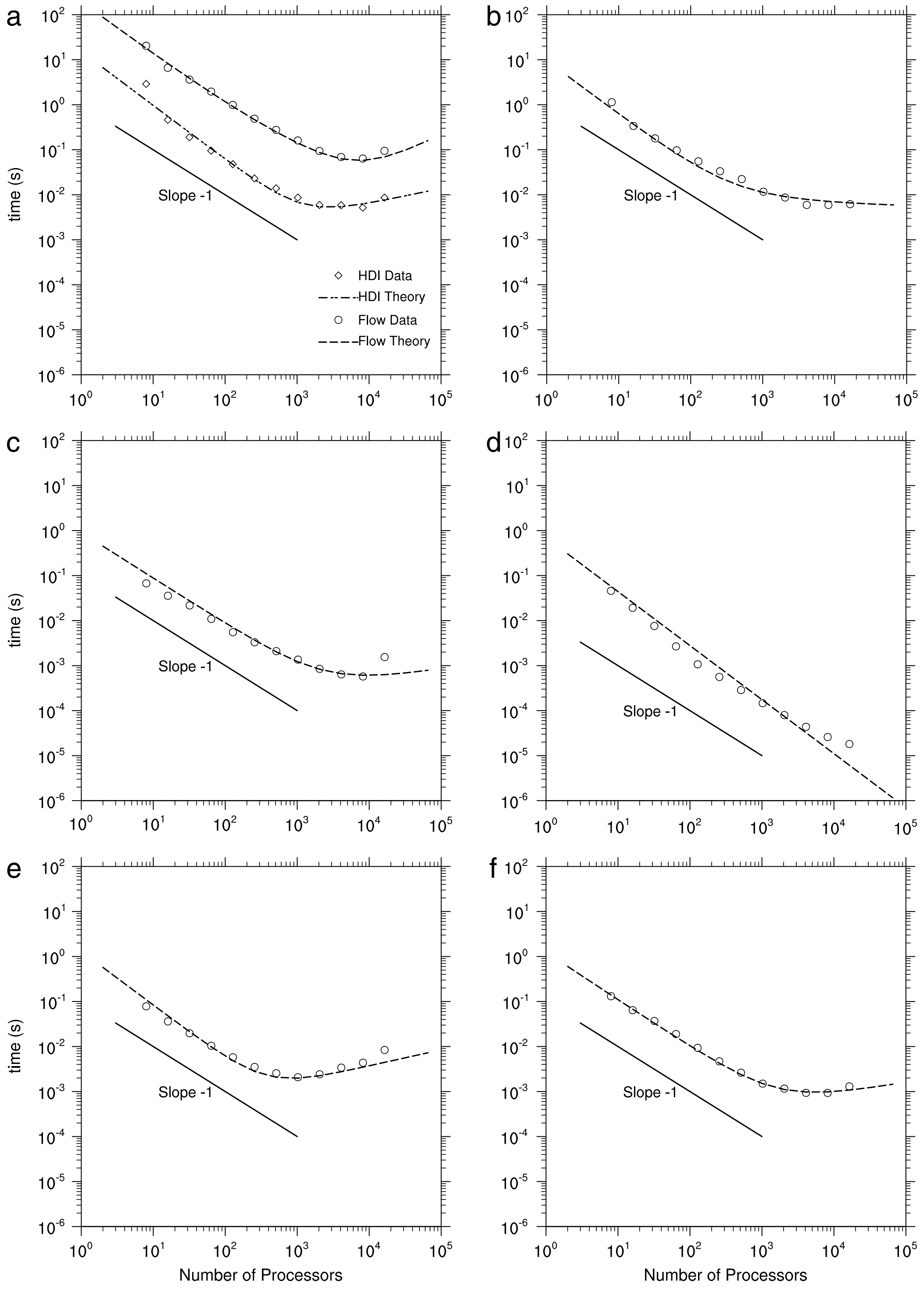


Fig. 5. Execution time as a function of P for a 512^3 DNS flow grid with 730,000 droplets of 45 and 50 μm size: (a) flow simulation and disturbance flow velocities, (b) interpolation, (c) cell-index and linked list method, (d) particle motion, (e) post-processing of droplet statistics, and (f) periodicity and non-overlapping conditions. The average number of iterations for the disturbance flow velocities for this case was 10. The lines correspond to the prediction from the complexity analysis and the markers denote the measured wall-clock time taken from Stampede.

(B.2d) and (B.2e), we note that the elemental times $t^{\text{comp}}_{(180N_p/P)}$ and $t^{\text{copy}}_{(180N_p/P)}$ reduce with P. Therefore when the elemental times multiply the N_p^2/P term, we obtain the better scaling than P^{-1}. This superscaling does not show for the flow calculations subroutine because the transmission time does not superscale and it plays a more important role in the whole performance of the FFT calculations.

4.2. Complexity analysis study

Except for the elemental times, the complexity analysis developed in the previous section has the advantage that it is independent of machine which allows the study of code performance in general, its dependence in problem setting and number of processors. For the sake

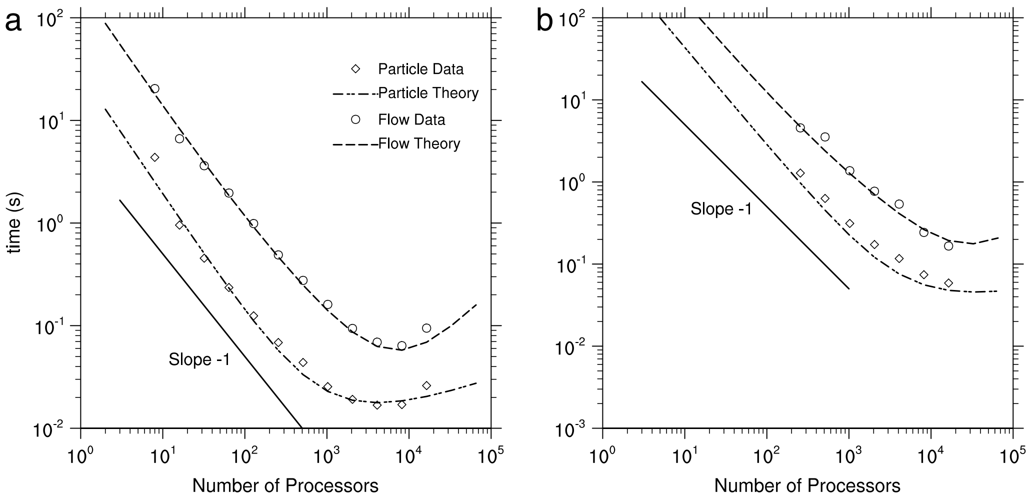


Fig. 6. Comparison between the complexity analysis and measured wall-clock time for two cloud microphysics settings: (a) 512^3, 45–50 μm, 730,000 particles (LWC $\sim$ 2.2 g/m^3), (b) 1024^3, 20–30 μm, 16,500,000 particles (LWC $\sim$ 1.0 g/m^3).

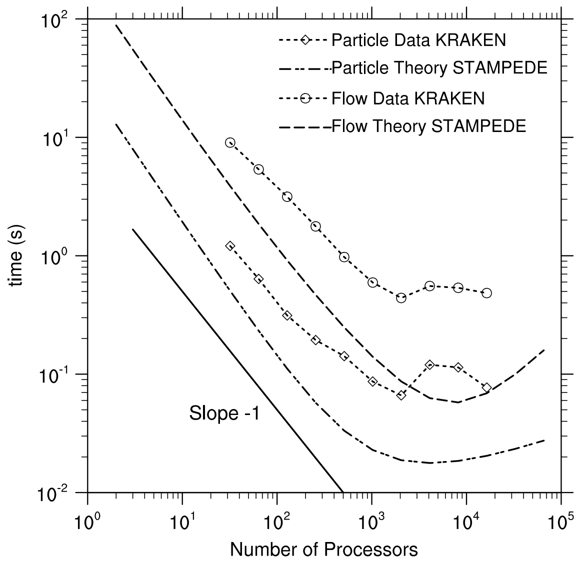


Fig. 7. Wall-clock time when running the code on Kraken for the case: 512^3, 45–50 μm, 730,000 particles (LWC $\sim$ 2.2 g/m^3). The complexity analysis of same case run in Stampede is also included for reference.

of simplification, we study code performance in the machines we have mentioned throughout the paper. Given that we have validated the complexity analysis against the observed wall clock times in the previous subsection, in Fig. 8 we apply the complexity analysis to study how the code performance changes with different parameter settings of the cloud microphysics problem. The typical liquid water content (LWC) in cloud is 1 g/m^3. In Fig. 8a we modify the LWC value and it can be seen that the LWC does not affect the wall-clock time significantly. Only for unrealistic values of LWC we could observe a noticeable increase in wall-clock time. The time do not increase linearly with LWC (or the total number of particle) because the most time consuming subroutine for the particles, the disturbance flow velocities subroutine, has an N_p^2 dependence (see Eqs. (B.1e) and (B.2e)). On the other hand, the wall clock time increases with the flow grid resolution, doubling the grid resolution results in an increase by a factor of about 8 (see Fig. 8b). When varying the particle radius (Fig. 8c), we noticed that for very small particles, the wall-clock time increases considerably because the number of particles being tracked increases notably to maintain the same LWC. In all these three tests, a monodisperse system of particles was considered. We also studied the effect of particle size a_2 when keeping a_1 at 25 μm, for a bidisperse system, in (Fig. 8d). For this system, we set $N_{p1} = N_{p2}$. There is a weak dependence of the wall-clock execution time with respect to the size of the particles in the second group. The reason is because the execution time is dominated by the flow calculations. The execution time will depend heavily on particle size if the size of all particles in the system is small.

In summary, the overall wall-clock time is dominated by the flow simulation task and depends strongly on the flow grid resolution, except for the cases when the particle radius becomes very small or the number of particles is very large. To study in more detail these issues, we compared in Fig. 9 the wall-clock time percentages for different subroutines using the complexity analysis. For the case of 4096^3 grid, a = 25 μm, and LWC = 1 g/m^3 in the left panel of Fig. 9, we confirm that the turbulent flow calculations are indeed the bottleneck of the code. The trends of all subroutines are similar to the ones observed in Fig. 4. In the right panel of same figure we show the case of 1024^3, a = 5 μm, and LWC = 1 g/m^3. In contrast to cases with larger particles as in the left panel, the bottleneck is the disturbance flow velocities calculations where the wall-clock time depends strongly on the number of particles. However, we find that the turbulent flow

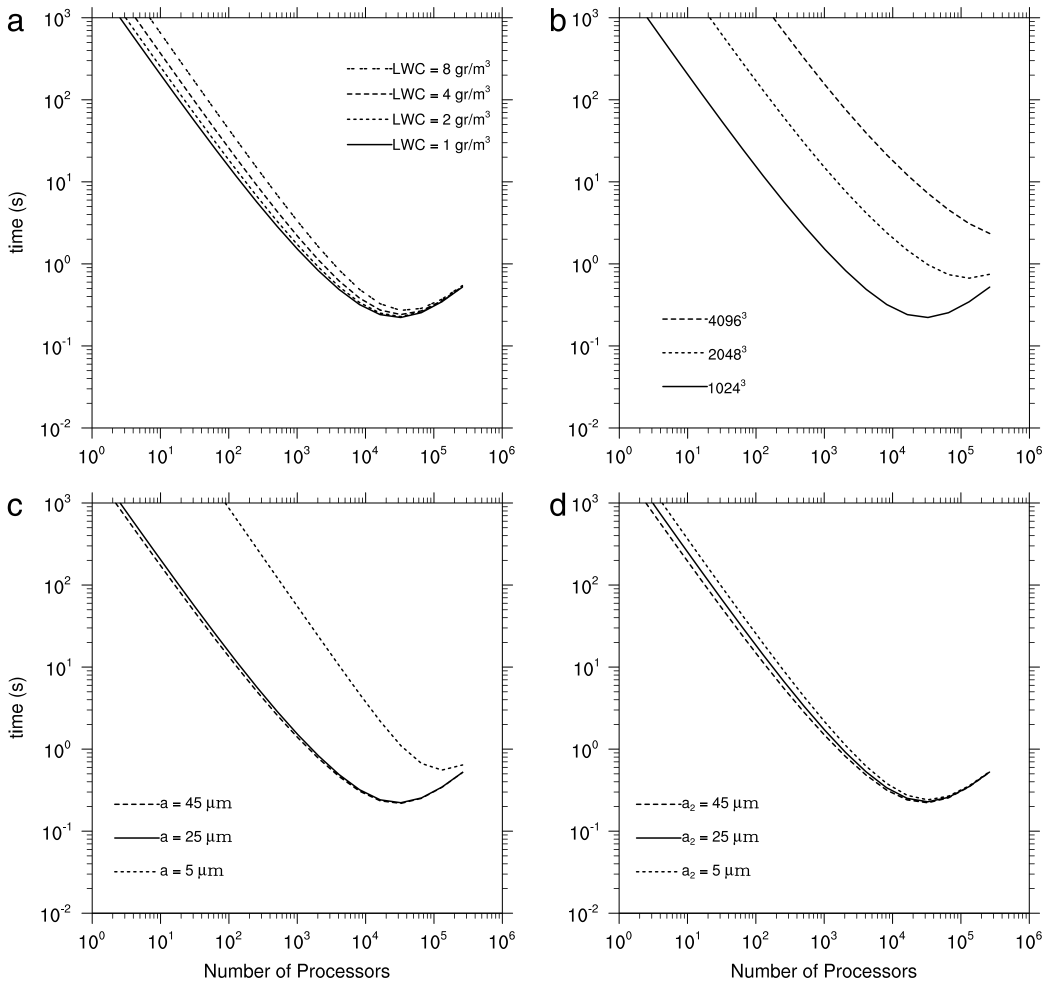


Fig. 8. Sensitivity of the wall-clock time on different cloud microphysics parameters. (a) Liquid water content (LWC), monodisperse system of a $= 25\ \mu$m, and 1024^3; (b) Turbulent flow resolution, monodisperse system of a $= 25\ \mu$m, and LWC $= 1$ g/m^3; (c) Particle size, monodisperse, 1024^3, and LWC $= 1$ g/m^3; and (d) Bidisperse case, $a_1 = 25\ \mu$ m, 1024^3, and LWC $= 1$ g/m^3.

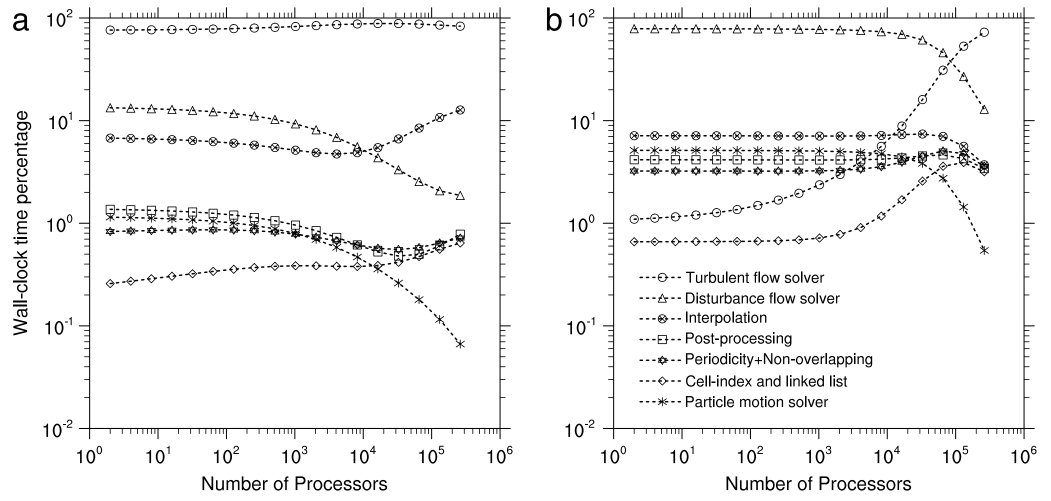


Fig. 9. Theoretical percentage of the wall-clock time per subroutine for two cloud microphysics cases from Fig. 8. (a) 4096^3, 25 μm (monodisperse), LWC $= 1$ g/m^3. (b) 1024^3, 5 μm (monodisperse), LWC $= 1$ g/m^3.

calculations increase its wall-clock time percentage as the number of processors is increased; due to the increasing communication latency. For a flow resolution of 1024^3, the maximum number of processors for the flow calculations should be 8196, therefore this subroutine will not become the bottleneck of the code for P less than 8196.

After comparing the times for different tasks for the extreme cases of cloud microphysics problems, we could state that either the turbulent flow calculations or the disturbance flow velocities could become the bottleneck. Therefore, they should be the focus for further

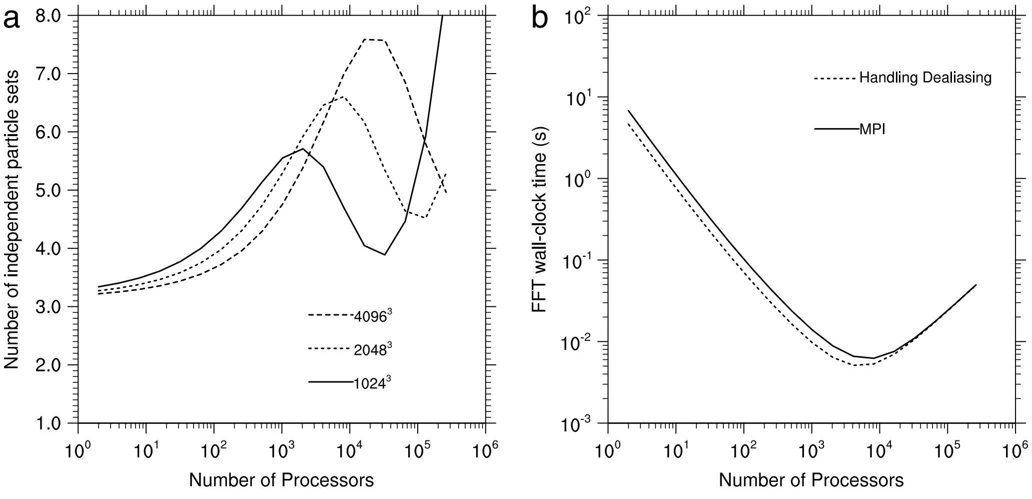


Fig. 10. Study of possible improvements to the code. (a) Number of parallel particle realizations to balance the flow simulation. Base case: 25 μm (monodisperse), LWC = 1 g/m^3; (b) FFT wall-clock time for the turbulent flow resolution at 512^3 grid.

improvements of the code. One option to increase the execution load for the particulate phase is to consider a number of independent sets of particles simulated at the same time (i.e., multiple particle realizations), such that the time to simulate the flow can be balanced by the time for handling the particulate phase. In Fig. 10a, we show this number for a base case of 25 μm (monodisperse), LWC = 1 g/m^3 and for different turbulent flow resolutions. For 1024^3 grid, we find that the number of independent sets of particles can be between 3 to 5 to balance the flow simulation. Note that the number of particle realizations or number of independent particles sets has a maximum and a minimum as P is increased. These points correspond to the number of processors at which the disturbance flow velocities and the FFT in turbulent flow calculations start to saturate and deteriorate. We note that the number of independent particle sets is obtained by the ratio of t_{flow} over $t_{particles}$. Now, as the number of processors increases, $t_{particle}$ decreases faster than t_{flow} (see Fig. 6), therefore the number of independent particle sets increases. At a certain number of processors the scaling of the execution time for particle calculations ($t_{particle}$) starts to deteriorate fast due to latency issues. This is when the curve starts to go down. Later it goes up again because t_{flow} starts to increase due to FFT latency issues.

There has been a few suggestions to improve the parallel performance of the pseudo-spectral simulation of turbulence in a periodic box. Practitioners in the field have proposed the use of: (1) communication with larger but fewer messages during the transpose operations for FFT, and (2) more efficient handling of the FFT making use of the fact that a significant portion of the domain in the spectral space is filled with zeros due to de-aliasing (see, for instance, [12]). As far as the first suggestion, Ayala and Wang [44] discussed the improvement by communicating larger but fewer messages during transpose operations for FFT. Plimpton at Sandia National Laboratories has presented another 2D decomposition strategy for complex-to-complex 3D FFTs (for details see Nelson et al. [50]). His strategy was to pack and send as much data as possible in a multistage algorithm using MPI_Send and MPI_Irecv commands. Ayala and Wang [44] found that this communication scheme performs better than our approach only for very large number of processors. As for the second suggestion, we shall discuss it making use of the complexity analysis. We argue that the observations from this analysis are independent of type of machine or interconnect. Such characteristics only modify the elemental times in the complexity analysis, thus they equally affect all scenarios.

In Fig. 10b, we show the execution time, based on the complexity analysis, of the FFT for 512^3 resolution. In addition to the current MPI implementation (Eq. (8)), we show the theoretical implementation based on the modification of the transpose operation in view of the zero values at large wave numbers in the spectral space. A full 3D FFT using 2D dd is composed by three FFTs, one real-to-complex (or complex-to-real) and two complex-to-complex 1D FFTs. For each group of 1D FFTs, a transpose of data is required. One scheme to avoid aliasing errors is to set to zero all flow velocity modes at wavenumbers $|k_x| > (2/3)N/2$, $|k_y| > (2/3)N/2$, and $|k_z| > (2/3)N/2$ (i.e., the 2/3 rule). In a 3D real-to-complex FFT, the first 1D FFT is real-to-complex and it is performed on the whole domain along the x-axis. All values with $|k_x| > (2/3)N/2$ will be set to zero. If so, only 2/3 of the domain is actually required to be transposed later to compute the other two complex-to-complex 1D FFTs (in y and z directions). Taking into consideration of the smaller relevant domain size, the complexity analysis for this second alternative is

$$t_{\text{FFT}}^{\text{Dealiasing}} = \frac{5}{2}\frac{N^3 \log_2 N^{7/3}}{P} t_{\text{FFT}}^{\text{comp}} + \frac{2}{3} 6 \frac{N^3}{P} t_{(N^3/P)}^{\text{copy}} + \frac{2}{3} 4 \frac{N^3}{P} t_{(P)}^{\text{comm}} + 4\left(\sqrt{P}-1\right) t_{(P)}^{\text{startup}}. \tag{16}$$

For the case shown in Fig. 10b, this alternative of taking advantage of zero modes shows an improvement of the speed up that does not exceed a factor of 1.5. The improvement is not as large as expected but it is not negligible for large simulations. Although not considered here, further improvements along this line may be possible. For instance, the two other complex-to-complex 1D FFTs could be modified in such a way that the zero modes for $|k_y| > (2/3)N/2$, and $|k_z| > (2/3)N/2$ can be considered. However, this might not lead to any gain because there might be at least one processor containing non-zero data in any given cycle within the transpose operation. Thus, the communication time for the busiest processor may not change for the other two complex-to-complex 1D FFTs which might lead to a negligible overall effect.

Let us now turn our attention to the case of tracking particles of small radius. As shown in the left panel of Fig. 9, the bottleneck in this case is the disturbance flow velocities calculations. The first question is how small the particle radius should be for the execution time of the particle section to become larger than the execution time of the flow section. The answer to this question is shown in Fig. 11a where the theoretical wall-clock time for the flow simulation and the particulate phase as a whole (t_{flow} and t_{particle}) are presented as a function of particle radius. The parameters for this monodisperse case are: 1024^3, LWC = 1 g/m^3, and P = 8196, although similar trends were

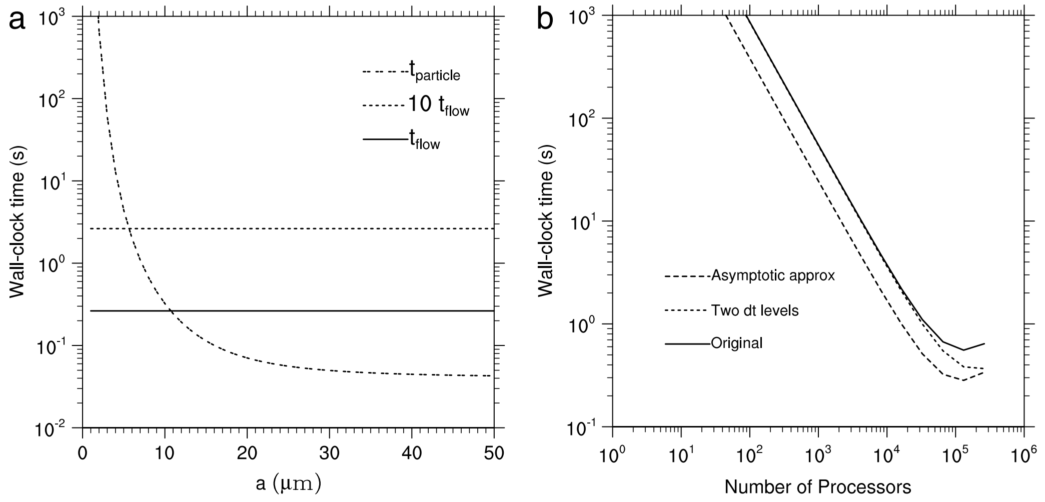


Fig. 11. Study of possible improvements to the code. (a) Theoretical wall-clock time for t_{flow} and t_{particle} to determine when $t_{\text{flow}} < t_{\text{particle}}$ (Case: 1024^3, LWC $= 1$ g/m^3, $P = 8196$). (b) Theoretical wall-clock time using two "dt" levels, and asymptotic approximation to track the particle, here d$t_{\text{flow}} >$ dt_{particle} (Case: 1024^3, 5 μm (monodisperse), LWC $= 1$ g/m^3).

observed for other combinations with the same LWC. For particles with $a < 5$ μm, the flow calculation time is negligible. This particle radius is obtained assuming an energy dissipation rate of 400 cm^2/s^3 used in this paper, and can change for other dissipation rates.

There is an additional issue concerning such small particles in terms of the time step size of the simulation. In most cases, the time step size to resolve the flow (dt_{flow}) is smaller than the time step size to resolve the particle trajectories ($dt_{\text{particles}}$). dt_{flow} is chosen such that the CFL is less than or equal to 0.25, while $dt_{\text{particles}}$ is $0.2\tau_p$. For small particles, $dt_{\text{particles}}$ could be smaller than dt_{flow}, implying that more integration steps are needed to cover a same physical time duration t_{final}. Therefore, the total execution time for the whole simulation is

$$T_{\text{exe}} = \left(t_{\text{flow}} + t_{\text{particle}}\right) \frac{t_{\text{final}}}{dt_{\text{particles}}}. \tag{17}$$

One possible approach to efficiently simulate the turbulent disperse flow is to allow two levels of time integration, namely, we could integrate the flow using a larger time steps than that used for the particle motion. Then, total execution time for the whole simulation is

$$T_{\text{exe}} = \left(t_{\text{flow}} \frac{dt_{\text{particles}}}{dt_{\text{flow}}} + t_{\text{particle}}\right) \frac{t_{\text{final}}}{dt_{\text{particles}}}. \tag{18}$$

An alternative approach to treat the motion of droplets at small Stokes numbers is to apply the asymptotic expansion of Maxey [51]. Since the stiff particle equation of motion is no longer integrated, we could use a significantly larger time step (i.e., dt_{flow} instead of $dt_{\text{particles}}$), yielding a better computational efficiency without much effect on the accuracy of the simulated collision statistics. However, due to the need to compute the fluid acceleration in the asymptotic expression for particle velocity, this alternative approach requires 3 additional FFTs, 3 additional interpolations and some extra memory usage. The total execution time is then

$$T_{\text{exe}} = \left(t_{\text{flow}} + 3\, t_{\text{FFT}} + t_{\text{particle}} + 3\, t_{\text{INTERP}}\right) \frac{t_{\text{final}}}{dt_{\text{flow}}}. \tag{19}$$

To estimate a priori the values of $dt_{\text{particles}}$ and dt_{flow}, we use $dt_{\text{particles}} = 0.2\tau_p$, and $dt_{\text{flow}} = 0.152N^{-0.43}\tau_k$ – an expression derived from our performed flow simulations with τ_k being the Kolmogorov time scale. Therefore,

$$\frac{dt_{\text{particles}}}{dt_{\text{flow}}} = 1.32N^{0.43}St. \tag{20}$$

At a turbulent dissipation rate of 400 cm^2/s^3, the Kolmogorov time scale is about 0.0206 s. Then, for cloud microphysics parameters, $dt_{\text{particles}}/dt_{\text{flow}} = 8.374 \times 10^{-4}N^{0.43}[a(\mu\text{ m})]^2$.

In Fig. 11b we show the theoretical estimated wall-clock time per time step of size $dt_{\text{particles}}$. We use the case of 1024^3, 5 μm (monodisperse), LWC $= 1$ g/m^3. As it can be noted, the approach of two dt levels does not improve the performance because t_{flow} is already smaller than t_{particle}. On the other hand, the approach using the asymptotic expansion improves the performance of the code with an increase of speedup by approximately 2.25.

Finally, the complexity analysis can be used to forecast the maximum number of processors our code could handle before the wall-clock execution time saturates and deteriorates due to communication latency issues. In Fig. 12 we show the maximum numbers of processors obtained from Eq. (14) and (15) that correspond to the "flow calculations" and "disturbance flow velocities" subroutines, respectively. We also include the maximum number of processor due to restrictions associated with the forcing implementation (P_{force}) and with the pair detection ($P_{\text{HI-detect}}$). An additional important information is the minimum number of processors capable to handle the problem due to memory requirements,

$$P_{\text{MEM}} = \frac{8 \times (10N^3 + 12N_p)}{1024^3 \times \text{MEMORY}^{\text{per-core}}_{\text{(Gbytes)}}}. \tag{21}$$

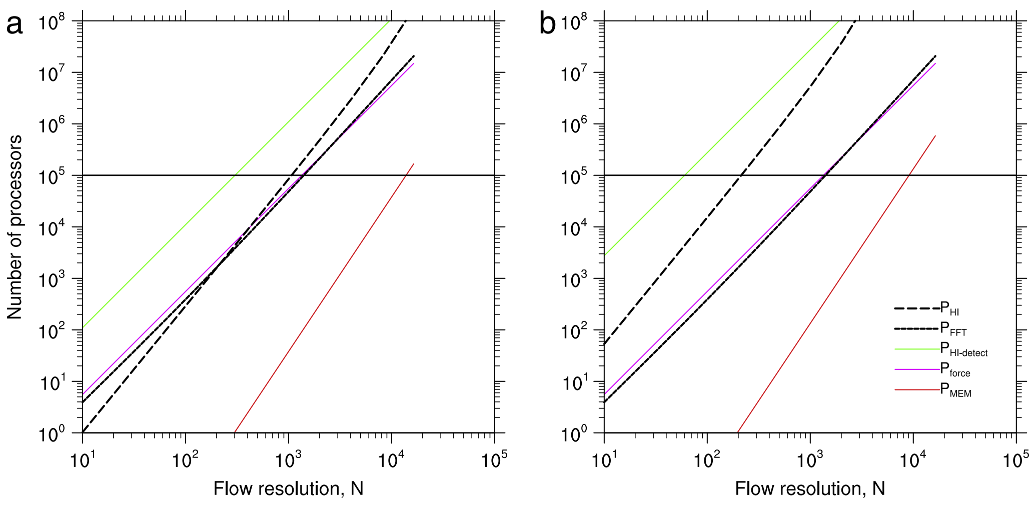


Fig. 12. Maximum (or minimum) number of processors allowed based on different considerations as a function of flow resolution. (a) Base case: 25 μm (monodisperse), LWC = 1 g/m^3; (b) Base case: 5 μm (monodisperse), LWC = 1 g/m^3.

In Stampede $\text{MEMORY}^{\text{per-core}}_{\text{(Gbytes)}} = 2$ Gbytes. The left panel of Fig. 12 corresponds to a monodisperse case of 25 μm particles. As the flow resolution increases, so is the maximum number of processors. When $P < 256$, the disturbance flow velocities saturates and deteriorates before the flow calculations. When $P > 256$, the flow calculations is the bottleneck of the code. Therefore, the maximum number of processors for such cases when the flow calculations are the bottleneck should be P_{FFT}. In contrast, in the right panel we show the case of very small particles when the disturbance flow velocities dominate the execution time. In this case, P_{HI} could be the maximum number of processors. However, the restriction due to large-scale forcing P_{force} pushes the maximum number of processor to P_{FFT}. Thus, it is safe to state that the maximum number of processors to run the code should be provided by P_{FFT} in general. The restriction due to pair detection is never violated. Another observation is that the minimum number of processors due to memory restrictions is not of concern. For Petascale machines (with number of processors $\mathcal{O}(10^5)$), we could efficiently solve problems up to 8192^3 grid resolution.

5. Application to turbulent collision of cloud droplets

The 2D dd implementation offers a unique opportunity to study sensitivity of the collision statistics of inertial particles on the range of scales resolved in DNS and equivalently on the flow Reynolds number. Using petascale computers, we were able to increase the scale separation in the flow and obtain collision statistics at a higher Taylor microscale Reynolds number ($R_\lambda \sim 400$). Simulations for even higher R_λ will be achievable with the use of larger grid resolutions and sufficiently large number of processors.

Quantitative evaluation of the effect of the larger scale turbulence on particle collision is essential to the fidelity of HDNS when applied to cloud microphysics. As a demonstration, here we focus on the R_λ-dependence of the kinematic collision kernel at flow dissipation of 400 cm^2/s^3.

So far, there have been rather limited simulation results of turbulent collision statistics at higher flow Reynolds numbers. Rosa et al. [52] showed kinematic collision statistics, i.e., radial distribution function and radial relative velocity of same-size droplets (of radius from 10 to 50 μm), for R_λ up to 204. The two pair statistics for monodisperse systems determine the collision kernel [35]. In a very recent study, Onishi et al. [26] have examined sensitivity of the collision statistics for a significantly wider range of R_λ (up to 527). They computed the flow with a fourth-order finite difference method. For the highest R_λ simulations, Onishi et al. used a grid of 2000^3 nodes and 10^9 particles. Results presented in [26] are limited to particles with Stokes number of 0.4. This corresponds to the cloud droplets with ∼25–30 μm in radius. Their results show that for such low Stokes numbers the R_λ-dependency of collision kernel is weak.

Here we simulate the kinematic collision kernel using highly accurate spectral code for a wide range of R_λ. To demonstrate the capability of the 2D dd implementation, we performed 8 sets of simulations for different droplet radii, i.e., 20, 25, 30, 35, 40, 45, 50 and 60 μm, and at different grid resolutions (from 32^3 up to 1024^3). The hydrodynamic interactions are not considered. The flow was forced by a stochastic forcing scheme [53]. The largest problem considered utilized a grid resolution of 1024^3 with 20 million particles and the time interval covered was around $8T_e$ (after the flow was stationary). Half of this time (around 4Te) was used to allow the droplets to settle into the stationary stage. Collision statistics were collected over the second half of the run time. Computations were performed on IBM Power 775 machine with 1024 processors.

Fig. 13 shows the resulting kinematic collision kernels as a function of R_λ. The collision kernel increases with droplet size due to both increased relative motion and local pair density of droplets. For a given droplet radius, the effect of large scales may be inferred from the changing R_λ. For droplets of radius 45 μm or less, the collision kernel increases with R_λ, but reaches a plateau for higher R_λ. The transition R_λ to the plateau increases with droplet size. For droplets larger than 45 μm in radius, the collision kernel increases monotonically with R_λ, and no saturation of the collision kernel is observed. This is expected as droplets of large radius have larger inertial response time and the relative pair statistics can be affected by a wider range of scales. Additional simulations with even greater Reynolds numbers are needed to study the asymptotic value of turbulent collision kernel for larger droplets. For a given size and flow Kolmogorov scales, R_λ affects both the particle clustering and radial relative velocity. Rosa et al. [53] showed that the radial relative velocity increases with R_λ when $R_\lambda < 84$ roughly (the actual transition value depends on droplet size), but becomes saturated for larger values. Therefore, the increase in the collision kernel for large R_λ is likely due to the dependence of clustering on the R_λ.

Also shown in Fig. 13 is the standard deviations of the time-averaged collision kernel, which are calculated following Rosa et al. [53]. The standard deviations represent statistical uncertainties and they are typically less than 1%. Achieving such precision was made possible by

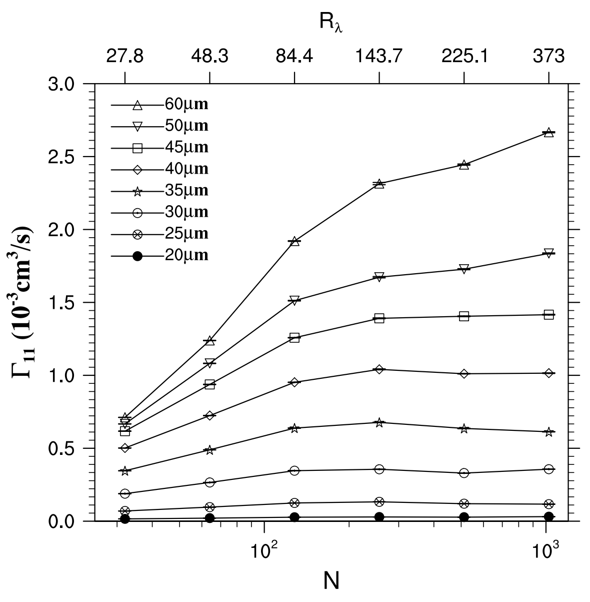


Fig. 13. Kinematic collision kernel of sedimenting droplets as a function of Re. Different lines correspond to simulations with different droplet radii.

tracking a large number of particles and employing highly accurate methods for computing radial distribution function and radial relative velocity. To compute these quantities we took advantage of their power-law scaling at short separation distances. The computation of the pair statistics at different separations requires intensive communication between processes. More details about this two-step approach are provided in Rosa et al. [53]. In the near future, we will study collision statistics for hydrodynamically-interacting droplets and their dependence on R_λ.

6. Summary and conclusions

In this paper, we have presented the details of a massively parallel implementation of turbulent particle-laden flow computation. The implementation is carried out in the context of cloud microphysics but could be extended to other applications depending on fluid and particle properties. We simulate the flow using a pseudo spectral DNS tool and incorporate the droplets which are being affected by Stokes drag and gravity and hydrodynamic interactions. The hydrodynamic interactions have been modeled by the Stokes disturbance flow approximation, see [39]. We studied the performance of the 2D dd scheme.

We developed a complexity analysis in order to predict the execution time for any problem size and any number of processors. This theoretical study helps to better understand the computational scalability of the method. The elemental times involved in this analysis are obtained empirically by fitting the actually measured elemental timing data. The resulting analysis was found to be in good agreement with the measured timing data for all the problem sizes and numbers of processors tested.

The execution time scales with number of processors almost linearly before the code saturates and deteriorates due to communication latency issues. The complexity analysis allowed to obtain an estimate of the maximum number of processors to sustain approximately this linear scalability, which shows that our code is likely to perform well on Petascale machines for large problem sizes.

We found that the turbulent flow calculations (using the pseudo spectral method) appear to be the bottleneck of the overall execution time taking about 80% of the whole computational time, for a dilute droplet suspension typical of cloud conditions. However, if the size of the particles is small ($a < 5$ μm for $\epsilon = 400$ cm^2/s^3, and keeping LWC $= 1$ g/m^3), the disturbance flow velocities become the bottleneck. We reviewed different suggestions to improve the performance of the turbulent flow calculations. We found that by handling appropriately the reduced flow domain size in the spectral space (due to the zeros to eliminate aliasing errors), we could speed up the flow calculations by only a factor of 1.50. In the case of small particle size, the asymptotic approximation approach to track the particles improves the performance of the code by speeding it up by a factor of 2.25.

The work presented here illustrates the feasibility of simulating particle laden turbulent flows on large computers using the pseudo-spectral method with direct integration of the equations of particle motion. In terms of physical results, preliminary results on the kinematic collision kernel for non-interacting sedimenting droplets show that for R_λ larger than 84 the collision kernel does not reveal R_λ-dependence for droplet sizes less than 40 μm. For larger droplets, the contribution of larger flow scales to the collision statistics is more dominant such that no saturation was observed. Therefore, additional simulations with even greater Reynolds numbers need to be performed in the future. In addition, it is necessary to study the kinematic collision kernel as a function of Reynolds number considering droplet–droplet hydrodynamic interactions.

Acknowledgments

This work was made possible by support from the National Science Foundation through grants OCI-0904534, ATM-0730766, and CRI-0958512 and by the National Center for Atmospheric Research (NCAR). NCAR is sponsored by the National Science Foundation. Computing resources in the US were provided by: (1) National Center for Atmospheric Research through CISL-35751010, CISL-35751014 and

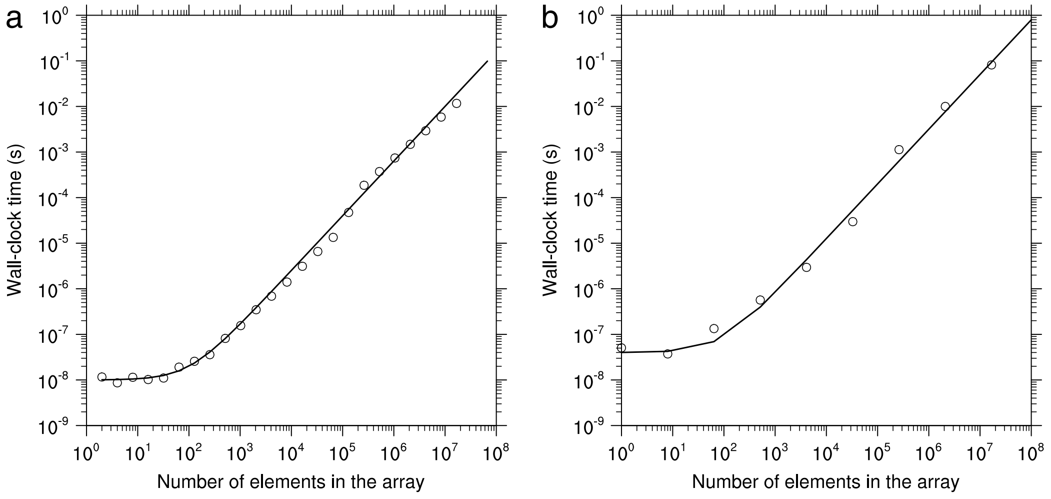


Fig. A.1. The measured computation and copy time as a function of array size (or number of elements in the array). The test runs were performed using double precisions. (a) Computational time; (b) Copy time. The lines correspond to the theoretical expressions and the markers correspond to actual measurements taken from Stampede.

CISL-35751015, and (2) Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575. We also express sincere gratitude to the Interdisciplinary Centre for Mathematical and Computational Modeling (ICM) at Warsaw University for providing computational resources (grant number G49-15).

Appendix A. Elemental times in the complexity analysis

Following similar procedure as in Ayala and Wang [44], the elemental times discussed in the complexity analysis can be estimated empirically by curve fitting execution times of some simple test runs on the designated machine (Stampede, in this case), or test runs of fully developed codes. The test runs were performed using $P_y \sim P_z$.

A.1. t^{comp} and t^{copy}

t^{comp} can be estimated by repeatedly doing some floating point operations. We carried out different array operations (multiplication, sum, exponential, sine, cosine, etc.) and average for a single overall value. The results are shown in Fig. A.1a. Note that for small number of elements in the operating array, the wall-clock time is constant. This time corresponds to the minimal time for retrieving data from one memory location to another. We propose and show in Fig. A.1a a theoretical expression

$$T_{\text{COMPUTATION}} = t^{\text{comp}}_{\text{Retrieval}} + t^{\text{comp}} N_{\text{FLOP}}, \tag{A.1}$$

where N_{FLOP} is the total floating point operations, and it is $N_{\text{FLOP}} = N_{\text{size}} \times FLOP$, where N_{size} is the size of the array. The time shown in Fig. A.1a is per FLOP (or, $FLOP = 1$). From the plot, we can extract:

$t^{\text{comp}}_{\text{Retrieval}} = 10$ ns

$t^{\text{comp}}_{(N_{\text{size}})} = 0.04 N_{\text{size}}^{0.20}$ ns/FLOP.

Ayala and Wang [44] also measured t^{comp} directly from their 3D FFT implementation. Following similar procedure, we estimated (neglecting retrieval time)

$t^{\text{comp}}_{(N_{\text{size}})} = 0.21$ ns/(word FLOP).

Interestingly, it does not depend on N_{size}.

Similarly, t^{copy} could be determined by running standard loops of memory-to-memory copy on a single processor and the results are shown in Fig. A.1b. The copy was performed following the Fortran order (i.e., copying along the first array direction, then the second, and lastly the third so the data is accessed contiguously in memory). As before, a theoretical estimate can be proposed as

$$T_{\text{COPY}} = t^{\text{copy}}_{\text{Retrieval}} + t^{\text{copy}} N_{\text{size}}. \tag{A.2}$$

From the plot, we can extract:

$t^{\text{copy}}_{\text{Retrieval}} = 40$ ns

$t^{\text{copy}}_{(N_{\text{size}})} = 0.20 N_{\text{size}}^{0.20}$ ns/word.

Measuring directly from Ayala and Wang [44] 3D FFT implementation, it can be estimated (neglecting retrieval time)

$t^{\text{copy}}_{(N_{\text{size}})} = 0.37 N_{\text{size}}^{0.20}$ ns/word.

It is important to mention that the elemental times have their complexity as they depend on many factors such as machine, compiler, computer programming language, array rank, array format (row-major or column-major formats, or also known as C order and Fortran order), number of FLOP in a code line, and number of processors. The experiments performed here with simple test runs and with a fully developed FFT code have their own characteristics in terms of array ranks and array formats which might not match the characteristics of other subroutines in our code. To achieve simplicity of the expressions, we only empirically adjust N_{size} to properly consider the differences.

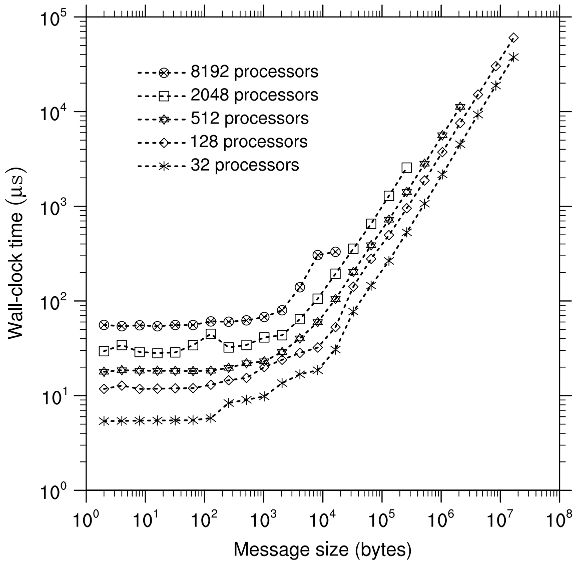


Fig. A.2. The communication time between a number of processors as a function of transmitted data size.

Table A.2
Estimated elemental times on Stampede for the complexity analysis.

$t^{\text{comp}}_{\text{Retrieval}}$	10 ns
$t^{\text{copy}}_{\text{Retrieval}}$	40 ns
$t^{\text{comp}}_{(N_{\text{size}})}$	$0.04N^{0.20}_{\text{size}}$ ns/FLOP 0.21 for FFT
$t^{\text{copy}}_{(N_{\text{size}})}$	$0.37N^{0.20}_{\text{size}}$ ns/word
$t^{\text{startup}}_{(P)}$	$0.83P^{0.27}$ μs
$t^{\text{comm}}_{(P)}$	$0.98P^{0.40}$ ns/word

A.2. t^{comm} and t^{startup}

The communication time is composed of the data transmission time and the latency time. They can be estimated by timing several simple communications (send and receive commands) using different array sizes between two processors in parallel with other processor pairs. The results of such experiments on Stampede are shown in Fig. A.2. The slope of the curves for large data sizes gives t^{comm} and the timing value in the limit of very small data size provides the latency time t^{startup}. As the message size decreases, a plateau indicates the latency time. The slope of each curve for large data size provides an estimate for t^{comm}. However, the plateau value and the slope changes with how busy the network of the machine is, therefore, both depend on the number of processors, P.

From the plot, we can extract:

$$t^{\text{startup}}_{(P)} = 0.83P^{0.40}\ \mu\text{s}$$
$$t^{\text{comm}}_{(P)} = 2.40P^{0.40}\ \text{ns/word}.$$

Measuring directly from Ayala and Wang [44] 3D FFT implementation, it can be estimated that

$$t^{\text{startup}}_{(P)} = 0.83P^{0.27}\ \mu\text{s}$$
$$t^{\text{comm}}_{(P)} = 0.98P^{0.40}\ \text{ns/word}.$$

We will adopt the elemental times estimated from the experiments with Ayala and Wang [44] 3D FFT implementation as they are more realistic from a fully developed code. However, since the FFT computational elemental time is a constant, we will use the one extracted from the simple experiments for the computations in other sections of the code. Thus, in summary, the elemental times for Stampede are given in Table A.2.

Appendix B. Complexity analysis of parallel calculations of disturbance flow velocities

We have recently developed a more efficient scheme to solve (3) based on the GMRes algorithm with preprocessing of coefficients [42]. This system of equations could be represented by $\mathbf{Ax} = \mathbf{b}$. The first phase of the subroutine is to compute the coefficients of the sparse matrix $\mathbf{A}$ and the vector $\mathbf{b}$ and to initialize the GMRes method. Eq. (B.1) accounts for the wall-clock time for this part of the subroutine

$$t_{\text{HDI}_{\text{Precomp}}} = \underbrace{124\, t^{\text{comp}}_{\text{Retrieval}} + 21\, t^{\text{copy}}_{\text{Retrieval}}}_{\text{Unparallelizable}} + \underbrace{2\, t^{\text{reduce}}_{(0.5,P)}}_{\text{Communicate global norm of }\mathbf{A}\text{ and }\mathbf{b}} \tag{B.1a}$$

$$+\underbrace{\left(28\,t^{\text{copy}}_{(180N_p/P)}+6\,t^{\text{comm}}_{(P)}\right)\left(\frac{50a_{\max}}{L\sqrt{P}}+2500\left(\frac{a_{\max}}{L}\right)^2\right)N_p+16\,t^{\text{startup}}_{(P)}}_{\text{Communicate halo regions with initial guess}} \tag{B.1b}$$

$$+\underbrace{\left[\left(62+2\left(3+\frac{50a_{\max}}{dx/2}\right)^3\right)t^{\text{comp}}_{(180N_p/P)}+\left(15+3\left(3+\frac{50a_{\max}}{dx/2}\right)^3\right)t^{\text{copy}}_{(180N_p/P)}\right]\frac{N_p}{P}}_{\text{Loop over all cells and particles}} \tag{B.1c}$$

$$+\underbrace{\left(18\,t^{\text{comp}}_{(180N_p/P)}+4\,t^{\text{copy}}_{(180N_p/P)}\right)\left(3+\frac{50a_{\max}}{dx/2}\right)^3\left(\frac{dx}{L}\right)^3\frac{N_p^2}{P}}_{\text{Search for pairs in respective cells}} \tag{B.1d}$$

$$+\underbrace{\left(48\,t^{\text{comp}}_{(180N_p/P)}+6\,t^{\text{copy}}_{(180N_p/P)}\right)\left(\frac{2.5\,10^5\pi}{3}\right)\left[\left(\frac{a_1}{L}\right)^3+\left(\frac{a_2}{L}\right)^3\right]\frac{N_p^2}{P}}_{\text{Precomputations of coefficients (only pairs in HI region)}}, \tag{B.1e}$$

where dx is the size of the cubic detection cell (note that $N_{cell} = L/dx$). As in the velocity interpolation subroutine, the data size for t^{comp} and t^{copy} is N_p/P, but in this case, there are also operations involving other array with higher ranks (such as 4 dimensional arrays) and in some cases, the loop over the arrays does not follow the Fortran order. In these cases, the performance for data copy and computation is affected, therefore an empirical factor of 180 was used. The term (B.1b) represents the communication to assign an initial value for the disturbance flow velocity felt by each particle in the halo regions. It has a structure similar to term (10b) for the HEAD and LIST formation, but with the main difference in the data size being sent.

The preprocessing of the coefficients also requires the pair search and this is measured by terms (B.1c) and (B.1d). Several comments are in order. In traditional coding of the cell-index method and the concept of linked lists, the search is only performed for the nearby 27 cells. Here we expanded the concept to search beyond the nearby cells to allow the number of cells N_{cell} (on each direction of the domain) be independent of the truncation radius ($50a$). The advantage of this is that the time for searching pairs for each detection cell, term (B.1d), remains small with decreasing dx or increasing the number of detection cells. This could minimize the cost for particle interactions, making it proportional to N_p rather than N_p^2. This was also shown recently by Onishi et al. [26]. However, the time for looping over all cells around each particle, in term (B.1c), increases with decreasing dx. For simplicity, let us assume that the copy time dominates the time for looping over all cells and the time for searching pairs at each cell. The optimal number of cells should happen when the wall-clock time of looping over all cells (Eq. (B.1c)) and searching pairs at each cell (Eq. (B.1d)) are similar. Using this logic, we could estimate that $N^{\text{optimal}}_{\text{cell}} \approx (1.33N_p)^{1/3}$, where the factor 0.75 is originated from the factor for copy in terms (B.1c) and (B.1d). Thus, it is interesting to recognize that the optimal number of detection cells depends on the number of particles and the number of copies for looping and searching for an individual cell. Onishi et al. [26] found that the optimal number of cells was at $N_p/N^3_{\text{cell}} \sim 1$, while our theoretical estimate yields $N_p/N^3_{\text{cell}} \sim 0.75$.

During the pair search, the preprocessing computation of the coefficients of **A** and **b** is represented through term (B.1e). This time depends on the truncation spherical volume where all droplets interacting with the chosen droplet are located. The GMRes algorithm requires the norm of **A** and **b**, which calls for two global communications among all processors using the MPI_ALLREDUCE command. The time for this reduction operation is denoted by the second term in (B.1a). In the subroutine there are a couple of unparallelizable operations, their execution time is represented in the first term in (B.1a).

Once the initialization and the preprocessing of coefficients are completed, the GMRes iterative process begins. Its execution time per iteration is estimated to be

$$\frac{t_{\text{HDI}_{\text{Iterat}}}}{n}=\underbrace{\left(\frac{851}{6}+\frac{11}{2}n+\frac{5}{3}n^2\right)t^{\text{comp}}_{\text{Retrieval}}+\left(\frac{33}{2}+\frac{3}{2}n\right)t^{\text{copy}}_{\text{Retrieval}}}_{\text{Unparallelizable}} \tag{B.2a}$$

$$+\underbrace{\left(\frac{3}{2}+\frac{n}{2}\right)t^{\text{reduce}}_{(0.5,P)}}_{\text{Communicate global norm of }\mathbf{A}} \tag{B.2b}$$

$$+\underbrace{\left(28\,t^{\text{copy}}_{(180N_p/P)}+6\,t^{\text{comm}}_{(P)}\right)\left(\frac{50a_{\max}}{L\sqrt{P}}+2500\left(\frac{a_{\max}}{L}\right)^2\right)N_p+16\,t^{\text{startup}}_{(P)}}_{\text{Communicate to update info from halo regions}} \tag{B.2c}$$

$$+\underbrace{\left[(20+9n)\,t^{\text{comp}}_{(180N_p/P)}+18\,t^{\text{copy}}_{(180N_p/P)}\right]\frac{N_p}{P}}_{\text{Computations over all particles}} \tag{B.2d}$$

$$+\underbrace{\left(19\,t^{\text{comp}}_{(180N_p/P)}+4\,t^{\text{copy}}_{(180N_p/P)}\right)\left(\frac{2.5\,10^5\pi}{3}\right)\left[\left(\frac{a_1}{L}\right)^3+\left(\frac{a_2}{L}\right)^3\right]\frac{N_p^2}{P}}_{\text{Computations using precomputed coefficients (only pairs in HI region)}}. \tag{B.2e}$$

The terms in this equation are similar to the terms in Eq. (B.1). The main differences are that the cell-index method and the concept of linked list is not used; there are additional computations per particle, term (B.2d); and certain terms have the number of iterations (n) to the second or third power due to the double nested loops in the GMRes algorithm. Torres et al. [42] found that the number of iterations depends on the problem setting, however they showed that it depends only weakly on N_p and droplet sizes (a_1, a_2). A typical iteration number is $\sim$10 for a convergence to 10^{-6} relative error. While it is possible to use other iterative schemes to reduce the data communication cost, they may not converge as quickly [42].

Appendix C. Complexity analysis of parallel calculations of post-processing droplet statistics

In this subroutine, we are interested in particle pair statistics which are typically more time consuming than single particle statistics. The execution time for this is estimated based on the dominant terms as

$$t_{\mathrm{POST}} = \underbrace{\left(28\, t^{\mathrm{copy}}_{\mathrm{Retrieval}} + 48\, t^{\mathrm{comm}}_{(P)}\right)\left(\frac{50a_{\max}}{L}\frac{1}{\sqrt{P}} + 2500\left(\frac{a_{\max}}{L}\right)^2\right)N_p + 16\, t^{\mathrm{startup}}_{(P)}}_{\text{Communicate to update info from halo regions}} \tag{C.1a}$$

$$+ \underbrace{\left[\left(24 + 23\left(3 + \frac{40a_{\max}}{dx}\right)^3\right)t^{\mathrm{comp}}_{(N_p/2P)} + \left(50 + \left(3 + \frac{40a_{\max}}{dx}\right)^3\right)t^{\mathrm{copy}}_{(N_p/2P)}\right]\frac{N_p}{P}}_{\text{Loop over all cells and particles}} \tag{C.1b}$$

$$+ \underbrace{13\, t^{\mathrm{reduce}}_{(2.0,P)}}_{\text{Collecting results}}\ . \tag{C.1c}$$

Most of the computations themselves were negligible and we do not show them. The code spends more time in collecting results and looping over all cells and particles for pair detection. We should point out that before the statistics are computed, some droplet data from halo regions have to be updated. Consequently, communications from halo regions are required, see term (C.1a). Let us recall that C_{size} for the elemental time of reduction communication is 0.5 (see introduction of Section 3) when the data size being transmitted is about 2 words. In this subroutine, the data size is larger than that, therefore C_{size} was chosen to be 2.0 to properly account for the difference.

References

[1] S. Elghobashi, Appl. Sci. Res. 48 (1991) 301–314.
[2] S. Elghobashi, Appl. Sci. Res. 52 (1994) 309–329.
[3] M. Sommerfeld, V.K.I. for Fluid Mechanics, Theoretical and Experimental Modelling of Particulate Flow, in: Lecture Series, No. 2000-6, 2000, pp. 1–62.
[4] L.P. Wang, B. Rosa, H. Gao, G.W. He, G.D. Jin, Int. J. Multiphase Flow 35 (2009) 854–867.
[5] G. Alfonsi, Appl. Mech. Rev. 64 (2011) 020802-1–020802-33.
[6] R.B. Pelz, J. Comput. Phys. 92 (1991) 296–312.
[7] E. Jackson, Z.S. She, S.A. Orszag, T. van, J. Sci. Comput. 6 (1991) 27–45.
[8] S. Chen, X. Shan, Comput. Phys. 6 (1992) 643–646.
[9] P. Mininni, D. Rosenberg, R. Reddy, A. Pouquet, Parallel Comput. 37 (2011) 316–326.
[10] D.A. Donzis, P.K. Yeung, D. Pekurovksy, Turbulence simulations at O(10^4) core counts, TeraGrid G08 Conference, Las Vegas, NV, 2008, (Science track paper).
[11] K. Itakura, A. Uno, M. Yokokawa, T. Ishihara, Y. Kaneda, Parallel Comput. 30 (2004) 1329–1343.
[12] P.K. Yeung, D.A. Donzis, K.R. Sreenivasan, B.L. Sawford, S.B. Pope, APS Meeting (2010) http://meetings.aps.org/link/BAPS.2010.DFD.RB.2.
[13] V. Heuveline, M.J. Krause, J. Latt, Comput. Math. Appl. 58 (2009) 1071–1080.
[14] H. Gao, H. Li, L.P. Wang, Comput. Math. Appl. 65 (2013) 194–210. http://dx.doi.org/10.1016/j.camwa.2011.06.028.
[15] K. Nakajima, Appl. Numer. Math. 54 (2005) 237–255.
[16] A. Gorobets, F.X. Trias, R. Borrell, O. Lehmkuhl, A. Oliva, Comput. & Fluids 49 (2011) 101–109.
[17] H. Enwald, E. Peirano, A.E. Almstedt, B. Leckner, Chem. Eng. Sci. 54 (1999) 311–328.
[18] H. Lindborg, V. Eide, S. Unger, S.T. Henriksen, H.A. Jakobsen, Comput. Chem. Eng. 28 (2004) 1585–1597.
[19] R. Onishi, Y. Baba, K. Takahashi, J. Computational Physics 230 (2011) 4088–4099.
[20] D. Darmana, N.G. Deen, J.A.M. Kuipers, J. Comput. Phys. 220 (2006) 216–248.
[21] B.P.B. Hoomans, J.A.M. Kuipers, W.J. Briels, W.P.M. van Swaaij, Chem. Eng. Sci. 51 (1996) 99–118.
[22] W.M. Charles, E. van den Berg, H.X. Lin, A.W. Heemink, M. Verlaan, J. Parallel Distrib. Comput. 68 (2008) 717–728.
[23] D.M. Beazley, P.S. Lomdahl, N. Gronbech-Jansen, R. Giles, P. Tomayo, Annual Reviews of Computational Physics III, World Scientific, Singapore, 1996, pp. 119–175.
[24] M. Winkel, R. Speck, H. Hubner, L. Arnold, R. Krause, P. Gibbon, Comput. Phys. Comm. 183 (2012) 880–889.
[25] K. Boryczko, W. Dzwinel, D.A. Yuen, Concurrency, Comput. Pract. Exp. 14 (2002) 137–161. http://dx.doi.org/10.1002i/cpe.619.
[26] R. Onishi, K. Takahashi, J.C. Vassilicos, J. Comput. Phys. (2013) http://dx.doi.org/10.1016/j.jcp.2013.02.027.
[27] R. Capuzzo-Dolcetta, M. Spera, D. Punzo, J. Comput. Phys. 236 (2013) 580–593.
[28] Th. Frank, H. Schneider, K. Bernert, K. Pachler, Trans. ASME J 125 (2003) 1–6.
[29] P. Pepiot, O. Desjardins, Powder Technol. 220 (2012) 104–121.
[30] B. Nkonga, P. Charrier, Parallel Comput. 28 (2002) 369–398.
[31] E. Bodenschatz, S.P. Malinowski, R.A. Shaw, F. Stratmann, Science 327 (2010) 970–971.
[32] H.R. Pruppacher, J.D. Klett, Microphysics of Clouds and Precipitation, Klumer Academic Publishers, AH Dordrecht, The Netherlands, 1997.
[33] R.A. Shaw, Annu. Rev. Fluid Mech. 35 (2003) 183–227.
[34] L.P. Wang, W.W. Grabowski, Atmos. Sci. Lett. 10 (2009) 1–8.
[35] W.W. Grabowski, L.P Wang, Annu. Rev. Fluid Mech. 45 (2013) 293–324. http://dx.doi.org/10.1146/1146/annurev-fluid-011212-140750. (an invited review paper).
[36] O. Ayala, B. Rosa, L.P. Wang, W.W. Grabowski, New J. Phys. 10 (2008) 075015.
[37] C.N. Franklin, P.A. Vaillancourt, M.K. Yau, J. Atmospheric Sci. 64 (2007) 938–954.
[38] L.P. Wang, O. Ayala, B. Rosa, W.W. Grabowski, New J. Phys. 10 (2008) 075013.
[39] L.P. Wang, O. Ayala, W.W. Grabowski, J. Atmospheric Sci. 62 (2005) 1255–1266.
[40] O. Ayala, W.W. Grabowski, L.P. Wang, J. Comput. Phys. 225 (2007) 51–73.

[41] B. Rosa, L.P. Wang, Lecture Notes in Comput. Sci. 6068 (2010) 388–397.
[42] C.E. Torres, H. Parishani, O. Ayala, L. Rossi, L.P. Wang, J. Comput. Phys. (2013) http://dx.doi.org/10.1016/j.jcp.2013.03.008.
[43] M.P. Allen, D.J. Tildesley, Computer Simulation of Liquids, Oxford University Press, 1987.
[44] O. Ayala, L.P. Wang, Parallel Comput. 39 (2013) 58–77.
[45] L.P. Wang, M.R. Maxey, J. Fluid Mech. 256 (1993) 27–68.
[46] L.P. Wang, A.S. Wexler, Y. Zhou, J. Fluid Mechanics 415 (2000) 117–153.
[47] Y. Zhou, A.S. Wexler, L.P. Wang, J. Fluid Mechanics 433 (2001) 77–104.
[48] S. Balachandar, M.R. Maxey, J. Comput. Phys. 83 (1989) 96–125.
[49] M.A.T. van Hinsberg, J.H.M. ten Thije, F. Toschi, H.J.H. Clercx, Phys. Rev. E 87 (4) (2013) 043307.
[50] J.S. Nelson, S.J. Plimpton, M.P. Sears, Phys. Rev. B 47 (1993) 1765–1774.
[51] M.R. Maxey, J. Fluid Mech. 174 (1987) 441–465.
[52] B. Rosa, H. Parishani, O. Ayala, L.P. Wang, W. Grabowski, Lecture Notes in Comput. Sci. 7204 (2012) 401–410.
[53] B. Rosa, H. Parishani, O. Ayala, W. Grabowski, L.P. Wang, New J. Phys. 15 (2013) 045032.