Parallel implementation of particle tracking and collision in a turbulent flow

Bogdan Rosa[1] and Lian-Ping Wang[2]

[1] Institute of Meteorology and Water Management,
ul. Podlesna 61, 01-673 Warsaw, Poland
bogdan.rosa@imgw.pl,
[2] Department of Mechanical Engineering, 126 Spencer Laboratory,
University of Delaware, Newark, Delaware 19716-3140, USA
lwang@udel.edu,
WWW home page: http://www.me.udel.edu/∼lwang/

Abstract. Parallel algorithms for particle tracking are central to the modeling of a wide range of physical processes including cloud formation, spray combustion, flows of ash from wildfires and reactions in nuclear systems. Here we focus on tracking the motion of cloud droplets with radii in the range from 10 to 60 μm that are suspended in a turbulent flow field. The gravity and droplet inertia are simultaneously considered. Our codes for turbulent flow and droplet motion are fully parallelized in MPI (message passing interface), allowing efficient computation of dynamic and kinematic properties of a polydisperse suspension with more than 10^7 droplets. Previous direct numerical simulations (DNS) of turbulent collision, due to their numerical complexity, are typically limited to small Taylor microscale flow Reynolds numbers (~ 100), or equivalently to a small physical domain size at a given flow dissipation rate in a turbulent cloud. The difficulty lies in the necessity to treat simultaneously a field representation of the turbulent flow and free movement of particles. We demonstrate here how the particle tracking and collision can be handled within the framework of a specific domain decomposition. Our newly developed MPI code can be run on computers with distributed memory and as such can take full advantage of available computational resources. We discuss scalability of five major computational tasks in our code: collision detection, advancing particle position, fluid velocity interpolation at particle location, implementation of the periodic boundary condition, using up to 128 CPUs. In most tested cases we achieved parallel efficiency above 100 %, due to a reduction in effective memory usage. Finally, our MPI results of pair statistics are validated against a previous OpenMP implementation.

1 Introduction

Air turbulence plays an important role in warm rain development. The process has been extensively investigated in many scientific studies [1, 2, 3], but complete quantitative understanding is still insufficient. The general description of

the multiscale interaction of cloud droplets with turbulent air flow is a challenging task due to inherent nonlinearities, inhomogeneities and coupling over disparate length and time scales. One of the main tools being used for cloud microphysics study is direct numerical simulation (DNS). Numerical complexity limits the DNS of turbulent collision to small Taylor microscale Reynolds number (R_λ ∼100). In real clouds the value is a few orders of magnitude larger ($R_\lambda \sim 10^4$). Achieving such high Reynolds numbers in numerical simulation is not feasible, but fortunately small flow structures are mainly responsible for droplets collision thus a truncated representation of turbulence is useful. Here, we intend to bring the numerical modeling closer to the physical conditions by extending the size of the computational domain. Serial codes or codes parallelized in OpenMP (e.g. [4]) for particle tracking can be run only on computers with shared memory where computational resources are limited to a single node. This restricts the grid resolution typically below 256^3. To perform simulation of turbulent collision on 512^3 grid or higher, different method of parallelization has to be applied.

In this paper, we report on the development of DNS of turbulent collision with parallel MPI library so the resulting code can be run on platforms with a distributed memory. Such a treatment allows the use of a larger number of processors (up 1024), memory, and improved cache utilization, leading to a much better overall computational efficiency. Our basic strategy of parallelization is domain decomposition, similar to what was previously utilized by Homman *et al.* [5]. The whole computational domain is divided into thin slabs in one direction and number of slabs corresponds to number of processors used. The differences between our implementation and Homman's lie in velocity interpolation and computation of particle pair statistics. Our new code computes in parallel radial distribution function (RDF [6] - a measure of the deviation of density distribution from uniform distribution), pair relative velocity, and dynamic collision rate. The parallel efficiency of these computations will be examined here.

2 Methodology

2.1 Flow simulation

The usual pseudo spectral method is employed to perform DNS of a forced isotropic homogenous turbulent flow. The incompressible Navier-Stokes and the continuity equation:

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

$$\nabla \cdot \mathbf{U} = 0 \tag{2}$$

are solved in a periodic cubic box. The flow domain is discretized uniformly into N^3 grid points, where in this study N takes the value of 128, 256, or 512. ω and P denote the fluid vorticity and pressure, respectively. The random forcing term $\mathbf{f}\left(x,t\right)$ is nonzero only for very low wave numbers (*i.e.*, $k \leq 2.5$), providing an energy source to sustain the air turbulence. Further details can be found

in [7]. The main difference between the current and previous implementation is the implementation of FFT (fast fourier transform). FFT demands free access to data from every grid point in the whole computational domain. Domain decomposition limits access of a given process to data in a given subdomain. This difficulty was overcome by splitting the full 3D (three dimensional) FFT into a series of 2D FFTs, parallel matrix transposition and then 1D FFTs. The transposition step reorganizes data to facilitate 2D and 1D FFTs within each process. To minimize the transpose operation, the domain is decomposed into slabs in the k_z direction in the wavevector space, but along the $y-$direction in the physical space [8]. The kinetic energy from low wave numbers propagates to small scales until viscous dissipation becomes active, eventually a quasi-steady kinetic energy balance is reached and flow becomes statistically stationary. This typically takes about 4 to 5 eddy turnover times after a random initialization of the flow field.

2.2 Particle tracking

When the turbulent flow reaches the statistically stationary stage, we introduce particles at random positions with a uniform distribution. The random numbers for setting the initial particle location are generated only by one master process. Then, the master process sends position data for particles within a given subdomain to an appropriate process. Although, such a treatment is not parallel, it is usually fast and only needs to be performed once. For example, generating positions of 5 million particles and distributing them among 64 processors takes only 0.7 [s]. This method ensures a true random distribution in the whole computational domain as only one seed for the random number generator is needed.

Assuming that the particles are small in comparison with the Kolmogorov microscale and particle density is much larger than the fluid density, the following equation of motion [9]

$$\begin{cases} \frac{d\mathbf{V}(t)}{dt} = \frac{\mathbf{u}(\mathbf{Y},t)-\mathbf{V}(t)+\mathbf{W}}{\tau_p} \\ \frac{d\mathbf{Y}(t)}{dt} = \mathbf{V}(t) \end{cases} \tag{3}$$

is solved, where $\mathbf{V}(t)$ and $\mathbf{Y}(t)$ are the velocity and the centre position of particle, respectively, τ_p is the particle inertial response time, $\mathbf{u}$ is the fluid velocity at the particle location, $\mathbf{W} = \tau_p \mathbf{g}$ is the still-fluid terminal velocity, and $\mathbf{g}$ is gravitational acceleration. Equation 3 was solved by the 4th order Adams Bashford method. The initial particle velocity is set to the fluid velocity at the particle location plus the terminal velocity.

2.3 Velocity interpolation

To compute the drag force acting on a particle, the fluid velocity at the particle location has to be interpolated from the solved fluid velocity field on a regular grid. Several different interpolation techniques have been developed previously, some of the more popular ones are shown in Table 1. In our MPI code

Table 1. Available interpolation schemes for fluid velocity at particle position.

Yeung, P. K. and S. B. Pope 1989 [10]	Third order, thirteen-point fourth-order cubic spline
Balachandar S, Maxey MR, 1989 [11]	6-pt Lagrangian interpolation
Squires, K. D. and J. K. Eaton 1990 [12]	Tri-linear
Rovelstad et al. 1994 [13]	Spectral, Tri-linear, Cubic spline, Hermite - mathematically equivalent to tri-cubic
Rouson et al. 1997,2008 [14, 15]	Tri-linear
Bec J 2005 [16]	Spectral (few modes)
Franklin et al. 2005, 2007 [17, 18]	Tri-linear
Lekien and Marsden 2005 [19]	Tri-cubic (implementation no DNS)
Busse et al. 2007 [20]	Tri-cubic
Homann et al 2007a, 2007b [5, 21]	Tri-cubic an tri-linear

we implemented a Lagrangian interpolation scheme using 6 grid points in each spatial direction [11, 7]. This requires data communication between neighboring processes in order to make sure that the full stencil of grid-based fluid velocity is always available when a particle is located in the vicinity of subdomain boundaries.

2.4 Periodic boundary condition

Particles moving in the turbulent flow field constantly change their subdomains or leave the computational box. When a particle moves into a different subdomain, the dynamic data occupied on the old process have to be transferred to a new process. Since the displacement of a particle during a time step is small, the particle will travel only to a neighboring slab. When a particle leaves the computational domain, periodic boundary condition is used to place the particle into a proper new process. Periodicity is realized by adding or subtracting the length of the computational box size from the particle coordinates. The communication time for moving data through processes depends on machine architecture and the number of nodes employed. Setting up separate communication between processors for sending and receiving data for every single particle is time consuming. Instead, we first made copies of the data in temporary buffer and then transferred the whole buffer in one operation. The original data table for particles within a given process is updated by removing selected items associated with particles that have left the subdomain, adding new particles just entered, and then re-indexing the whole table.

2.5 Collision detection

The algorithm for collision detection implemented in the MPI code follows the idea presented by Wang et al. [22], with several modifications resulting from different memory management. Dynamic collision detection is executed in two steps. In the first step, collisions within a given subdomain are detected using an efficient linked list method [23]. In the second step the algorithm detects collisions

occurring between particles from two different processes. The second step is faster because searching is limited only to a narrow slice covering overlapping region between two neighboring processors. To perform this step, the complete set of data with particles location, velocity and particles size has to be transferred to a neighboring process. The above treatment retains full parallelism and minimizes amount of data which has to be sent and received between CPUs.

2.6 Radial distribution function and relative velocity

Radial distribution function is computed based on the definition proposed in [6]. Domain decomposition introduces additional boundaries inside the domain and complicates the detection of all particle pairs with separation distance in the range from $r - \delta$ to $r + \delta$. This problem was solved again by a two-step procedure, similar to that used in the collision detection. Namely, particle pairs inside any given subdomain are detected first, then additional pairs involving particles from different processes are found. Relative velocity is computed for every pair used for the RDF calculation. These kinematic statistics are further averaged over time.

3 Parallel performance

To examine the scalability of the MPI code, a number of numerical experiments were performed. All tests presented in this paper were conducted on an IBM Power 575 cluster (4064 POWER6 processors running at 4.7 GHz) at NCAR's supercomputing center.

In the first test we compare times designated for five major parts of the code: evolution of the turbulent flow, collision detection, velocity interpolation, advancing particle position and the implementation of the periodic boundary condition. The wall clock times using different number of processors and a given number of time steps are collected to determine the scalability for each of the tasks separately. The total measured time is the sum of computational and communication time. Additional time spent for saving data is negligible compared to either the computational or communication time. Maximal number of processors used in the test is 128 (4 full nodes). Figure 2 shows the total time needed for each task as a function of the number of processors.

The time spent for tasks related to the particle motion (collision detection, velocity interpolation, advancing particle position and periodic boundary condition) is inversely proportional to the number of processors. For the flow evolution the time also decreases but only for small number of processors (up to 32). For larger number of processors (64 and 128), the computational time appears to saturate, due to the increasing communication time associated with parallel FFT.

On the right panel of Fig. 1 we show the ratio of total computational time over the total execution (computational plus communication) time as a function of the number of processors. The computational time takes about half of the time for intermediate numbers of processors, but less than one third for larger numbers of processors. This shows that it is very critical to minimize the communication overhead in the MPI code.

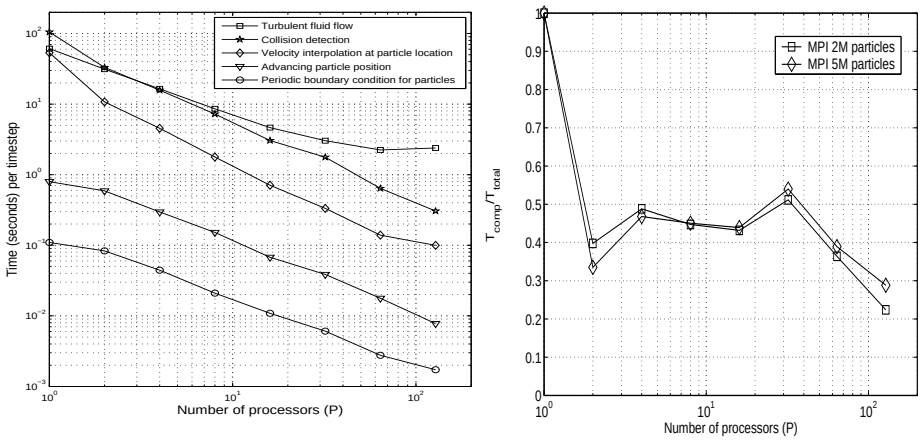

Fig. 1. Left panel: Scalability of the five major tasks in the DNS code in terms of the total execution time. Right panel: The ratio of $T_{computation}/T_{total}$ as a function of the number of processors. The grid size is 512^3.

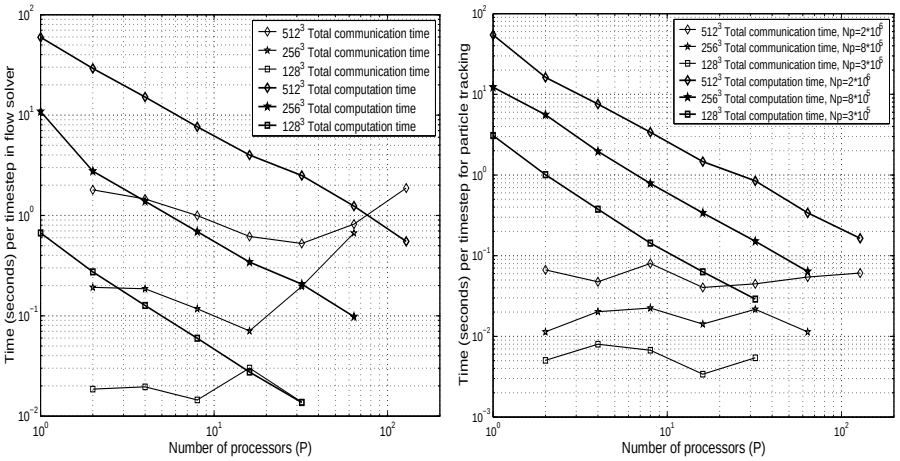

Fig. 2. Scalability of computational and comunication time for the flow solver (left panel) and the particle part (right panel), as a function of the number of processors.

In the second analysis, computational time and communication time for three different grid sizes 128^3, 256^3 and 512^3 are examined separately. Here the tasks are only divided into two groups: the flow solver part and particle part. Separate scaling performances are shown in figure 3. The computational time for both parts decreases roughly linearly with the number of processors, in the log-log plots. The dependence of the communication time on the number of processors is more complex. For the flow evolution part, a reduction of communication time is realized with an increase in the number of processors from 1 to 32. This

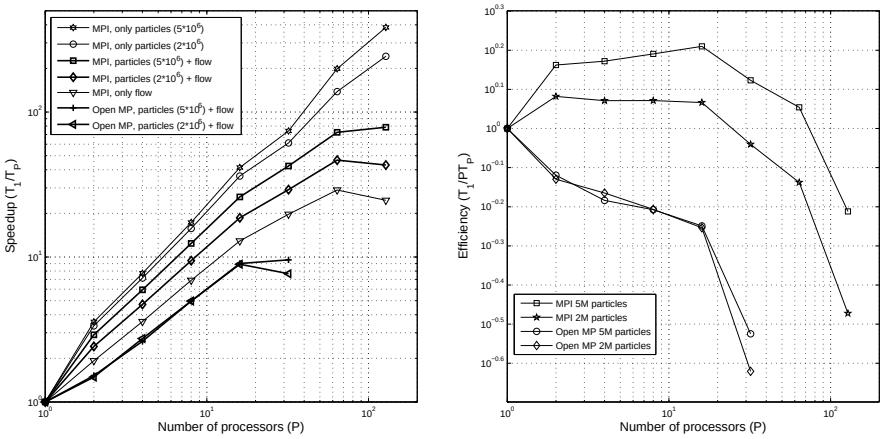

Fig. 3. Comparison of speedup (left panel) and computational efficiency (right panel) of the DNS codes parallelized with two different libraries (MPI and OpenMP). Grid is 512^3.

can be explained because amount of data being transferred between processes decreases with increasing number of processors. For larger number of processors more connections have to be established and the communication time in fact increases eventually with the number of processes. For the particle part the communication time is insensitive to the number of processors used.

In the third analysis, we compare the performance of two codes: our new MPI code and an OpenMP code developed previously by Ayala *et al.* [4]. Both codes are functionally the same and yield identical results of collision statistics. Here we compare the speedup, efficiency (fig. 4), and the memory usage (fig. 5) of the codes, under an identical setting (the same initial flow field, the same number of particles, the same initial particle location and the same particle size). The speedup presented in figure 4 is determined by simulating 2 and 5 million particles starting from the same particle location and the same flow velocity field. Results obtained with OpenMP code show that, for the maximum allowed number of processors (32 on a single node), the speedup cannot exceed 10. In contrast, the MPI code can be run on more than one node and as such can be run on any number of processes. Figure 4 shows that, for the MPI code, the highest achievable speedup for 2 million particles occurs at 64 processors while for 5 million particles the highest speedup is for 128 processors. We can conclude that the speedup depends on the number of particles tracked. For 20 million particles, we expect that more than 128 processors can be used to achieve an efficiency close to 100%. In the Open MP code, efficiency decreases monotonically with the number of processes and it does not exceed 30%.

The small difference between two MPI curves in Figure 5 shows that the majority of memory is used for flow solver rather than for the particle part. The memory usage for the particle part will not exceed 10 GB for 20 million particles. The high increase of memory usage between 32 processors and 128 is due to the

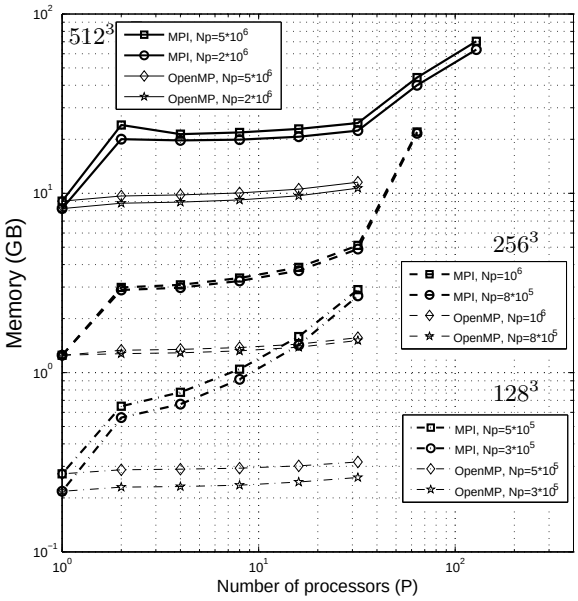

Fig. 4. Memory usage.

allocation of additional buffers needed for communication. In the MPI code the matrices with particle data was oversized by around 30 % in order to handle the particles which are moving between processes. Since in the Open MP code there is no need to allocate such oversized matrices and particle data size is defined precisely, memory usage is about 10 GB and is significantly less than in the MPI code.

Finally, we compare dynamic and kinematic collision kernels computed using the two codes (MPI and OpenMP). Figure 6 shows that the two codes give the same results and differences are within the statistical uncertainties of the data.

4 Conclusions

MPI implementation has been developed for a code designed to study turbulent collision of particles in a turbulent flow. This is a challenging task since turbulent flow and particle transport require different implementation strategies. Here we discussed major issues and implementation details, along with some ideas for optimizing the MPI code. A number of numerical tests are used to show scalability, speedup, and overall efficiency, and times required for different tasks are compared. The MPI code performs better than an earlier OpenMP code, and can take full advantage of distributed memory hybrid computers. Our next step is the MPI implementation for particle-particle aerodynamic interaction which requires considerations of both short and long-range interactions.

Acknowledgements

This work was supported by the National Science Foundation (NSF) under grants ATM-0527140 and NSF ATM-0730766. Computing resources are pro-

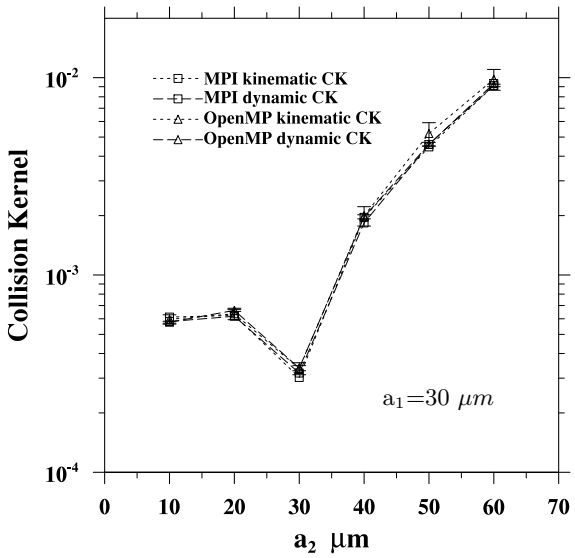

Fig. 5. Comparison of dynamic and kinematic collision kernels for sedimenting droplets in turbulent air computed in the MPI and OpenMP codes.

vided by National Center for Atmospheric Research (NCAR CISL-35751010).

References

[1] O. Ayala, B. Rosa, L. P. Wang, and W. W. Grabowski. Effects of turbulence on the geometric collision rate of sedimenting droplets. Part 1. Results from direct numerical simulation. *New Journal of Physics*, 10, 2008.

[2] O. Ayala, B. Rosa, and L. P. Wang. Effects of turbulence on the geometric collision rate of sedimenting droplets. Part 2. Theory and parameterization. *New Journal of Physics*, 10, 2008.

[3] L. R. Collins and A. Keswani. Reynolds number scaling of particle clustering in turbulent aerosols. *New Journal of Physics*, 6, 2004.

[4] Wang L-P. Ayala O., Grabowski W.W. A hybrid approach for simulating turbulent collisions of hydrodynamically-interacting particles. *JCP*, pages 51–73, 2007.

[5] H. Homann, J. Dreher, and R. Grauer. Impact of the floating-point precision and interpolation scheme on the results of DNS of turbulence by pseudo-spectral codes. *Computer Physics Communications*, 177(7):560–565, 2007.

[6] Zhou Yong Wang L-P., Wexler A. S. Statistical mechanical description and modelling of turbulent collision of inertial particles. *J. Fluid Mech.*, pages 117–153, 2000.

[7] L. P. Wang and M. R. Maxey. Settling velocity and concentration distribution of heavy-particles in homogeneous isotropic turbulence. *Journal of Fluid Mechanics*, 256:27–68, 1993.

[8] P. Dmitruk, L. P. Wang, W. H. Matthaeus, R. Zhang, and D. Seckel. Scalable parallel FFT for spectral simulations on a Beowulf cluster. *Parallel Computing*, 27(14):1921–1936, 2001.

[9] M. R. Maxey and J. J. Riley. Equation of motion for a small rigid sphere in a nonuniform flow. *Physics of Fluids*, 26(4):883–889, 1983.

[10] P. K. Yeung and S. B. Pope. Lagrangian statistics from direct numerical simulations of isotropic turbulence. *Journal of Fluid Mechanics*, 207:531–586, 1989.

[11] S. Balachandar and M. R. Maxey. Methods for evaluating fluid velocities in spectral simulations of turbulence. *JCP*, pages 96–125, 1989.

[12] K. D. Squires and J. K. Eaton. Particle response and turbulence modification in isotropic turbulence. *Physics of Fluids a-Fluid Dynamics*, 2(7):1191–1203, 1990.

[13] A. L. Rovelstad, R. A. Handler, and P. S. Bernard. The effect of interpolation errors on the lagrangian analysis of simulated turbulent channel flow. *Journal of Computational Physics*, 110(1):190–195, 1994.

[14] D. W. I. Rouson, S. C. Kassinos, I. Moulitsas, I. E. Sarris, and X. Xu. Dispersed-phase structural anisotropy in homogeneous magnetohydrodynamic turbulence at low magnetic Reynolds number. *Physics of Fluids*, 20(2):19, 2008.

[15] John K. Eaton Damian, W. I. Rouson and Scott D. Abrahamson. A direct numerical simulation of a particle-laden turbulent channel flow. *No. TSD-101*, 1997.

[16] J. Bec. Multifractal concentrations of inertial particles in smooth random flows. *Journal of Fluid Mechanics*, 528:255–277, 2005.

[17] C. N. Franklin, P. A. Vaillancourt, M. K. Yau, and P. Bartello. Collision rates of cloud droplets in turbulent flow. *Journal of the Atmospheric Sciences*, 62(7):2451–2466, 2005.

[18] C. N. Franklin, P. A. Vaillancourt, and M. K. Yau. Statistics and parameterizations of the effect of turbulence on the geometric collision kernel of cloud droplets. *Journal of the Atmospheric Sciences*, 64(3):938–954, 2007.

[19] F. Lekien and J. Marsden. Tricubic interpolation in three dimensions. *International Journal for Numerical Methods in Engineering*, 63(3):455–471, 2005.

[20] A. Busse, W. C. Muller, H. Homann, and R. Grauer. Statistics of passive tracers in three-dimensional magnetohydrodynamic turbulence. *Physics of Plasmas*, 14(12), 2007.

[21] H. Homann, R. Grauer, A. Busse, and W. Mueller. Lagrangian statistics of Navier-Stokes and MHD turbulence. *Journal of Plasma Physics*, 73:821–830, 2007.

[22] Ayala O. Wang L-P., Franklin C. N. and Grabowski W.W. Probability distributions of angle of approach and relative velocity for colliding droplets in a turbulent flow. *JAS*, pages 881–900, 2006.

[23] M. P. Allen and D. J. Tildesley. Computer simulation of liquids. *New York: Oxford University Press*, page 408, 1987.