International Journal of Multiphase Flow 99 (2018) 437–445

Contents lists available at ScienceDirect

International Journal of Multiphase Flow

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

GPU acceleration of Eulerian–Lagrangian particle-laden turbulent flow simulations

James Sweet[b], David H. Richter[a,*], Douglas Thain[b]

[a] University of Notre Dame, Department of Civil and Environmental Engineering and Earth Sciences, Notre Dame, IN 46556, United States
[b] University of Notre Dame, Department of Computer Science and Engineering, Notre Dame, IN 46556, United States

ARTICLE INFO

Article history:
Received 8 September 2017
Revised 8 November 2017
Accepted 10 November 2017
Available online 11 November 2017

Keywords:
Graphics processing unit
Particle-laden turbulent flow simulation

ABSTRACT

Acceleration of an existing MPI-based, particle-laden turbulent flow simulation code is achieved using up to four NVIDIA GPU devices. The overall design is to transfer the entire flow velocity, temperature, and humidity fields to each device, and compute particle trajectories entirely on the GPU hardware. For one-way coupled turbulent flow simulations, accurate simulations can be achieved for less computational cost than the original CPU implementation for particle numbers above 10^6. Above 10^7 particles, the GPU version is roughly 14 times faster than the original CPU implementation. The effects of interpolation order, precision of the transferred Eulerian fields, and sub-time-stepping for fast particle dynamics are discussed.

1. Introduction

Computational acceleration via Graphics Processing Units (GPUs) has increasingly become more common in recent years, as overall hardware performance and knowledge base has increased. Potential gains in computational speed, overall power efficiency, and comparable hardware cost continue to make scientific computing with GPUs an appealing option for many scientific fields.

One such field of study is in computational fluid mechanics, which centers around integrating the Navier–Stokes equations to produce realizations of various fluid flows. Within this broad umbrella, GPU acceleration typically takes one of three forms. First, the entire flow solver can be put onto one or more GPUs. Salvadore et al. (2013) for instance present accurate simulations of a turbulent flow using a single NVIDIA GPU device, having ported and optimized an existing Fortran computational code to the NVIDIA CUDA (Compute Unified Device Architecture) language. Schalkwijk et al. (2012) and Schalkwijk et al. (2015) demonstrate that large eddy simulation (LES) of the planetary boundary layer can be massively accelerated when porting existing codes to multiple GPUs, enabling simulations previously thought inaccessible, such as the use of LES as a weather prediction tool. Many other implementations exist (van Heerwaarden et al., 2017; Khajeh-Saeed and Perot, 2013; Manuel et al., 2014), and issues associated with optimization and porting continue to be studied (Aissa et al., 2017).

A second technique for accelerating flow solvers via GPU is within the class of particle-based methods, including Lattice-Boltzmann methods (LBM) and smoothed particle hydrodynamics (SPH). Here, the implementation on GPU is arguably simpler than porting Eulerian-based methods, since tracking large numbers of particles is an operation which better exploits the GPU parallelization. GPU-accelerated SPH and LBM methods are used for a variety of applications (Mokos et al., 2017; Obrecht et al., 2013; Winkler et al., 2017), and are oftentimes a powerful means of solving complex flows on simple desktop systems (Gomez-Gesteira et al., 2012; Hérault, 2010). The third class of GPU-based flow solvers focuses on accelerating only certain expensive and parallelizable sections of the overall solver. This can be quite specific, such as physics-based radiation schemes in weather prediction models (Michalakes and Vachharajani, 2008; Mielikainen et al., 2012), or via acceleration of linear algebra packages for solving large linear systems (Minden et al., 2013; Tomov et al., 2010).

The present work falls more into the latter category, as the type of problem we aim to enhance with GPUs is particle-laden simulations of turbulent flows. More specifically, we aim to accelerate the particle calculation of the common Eulerian-Lagrangian representation of dispersed phase turbulent flows. In this approach, a grid-based direct numerical simulation (DNS) or LES is coupled to a point-particle treatment of the particle phase, where each particle is integrated individually (Balachandar and Eaton, 2010). It is this individual treatment of the particles which allows for straightforward porting with significant speedup.

* Corresponding author.
E-mail address: david.richter.26@nd.edu (D.H. Richter).

https://doi.org/10.1016/j.ijmultiphaseflow.2017.11.010
0301-9322/© 2017 Elsevier Ltd. All rights reserved.

As detailed below, the overall strategy is to offload the entire particle computation to one or more GPUs, which requires transferring gridded flow velocity, temperature, and humidity data from the CPU cluster to the GPU devices. As such, we note that this work is meant to target relatively modest flow calculations that can benefit from many more particles than traditional MPI (Message Passing Interface) implementations can provide, thereby enabling larger ensembles of particles for faster statistical convergence or higher resolution of Lagrangian-based continuous fields. It is shown that for numbers of particles of order 10^8, the use of up to four NVIDIA GPU devices can produce significant speedup and allow for multiresolution treatment of fast particle processes that would otherwise be computationally expensive.

2. Algorithm and code structure

In this section, first the original implementation will be outlined, followed by a description of how GPU hardware is used to accelerate the particle calculations. In describing the GPU implementation, we focus on three key factors which reflect tradeoffs between performance and accuracy: interpolation of flow quantities to particle locations, single versus double precision considerations, and introducing multiresolution sub-time-stepping for resolving particle time scales shorter than the flow time step.

2.1. Existing code

The original code is a modified version of the National Center for Atmospheric Research (NCAR) LES model (Moeng, 1984; Sullivan et al., 1994), extended to include Lagrangian particles (here referred to as the NTLP code — NCAR Turbulence with Lagrangian Particles). The NTLP code has recently been used in numerous DNS studies to investigate the dynamics of inertial, non-isothermal, and/or evaporating particles in various settings (Gonzalez et al., 2017; Helgans and Richter, 2016; Peng and Richter, 2017; Richter et al., 2016; Richter and Sullivan, 2013; 2014). For the Eulerian flow calculation, the code solves the incompressible Navier-Stokes equations for the velocity, temperature, humidity, and pressure fields on a fixed Cartesian mesh:

$$\nabla \cdot \mathbf{u} = 0, \tag{1}$$

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho}\nabla p + \nabla \cdot (\nu \nabla \mathbf{u}), \tag{2}$$

$$\frac{\partial T}{\partial t} + \nabla \cdot (\mathbf{u}T) = \nabla \cdot (\alpha \nabla T), \tag{3}$$

$$\frac{\partial q}{\partial t} + \nabla \cdot (\mathbf{u}q) = \nabla \cdot (\Gamma \nabla q), \tag{4}$$

where $\mathbf{u}$ is the fluid velocity, p is the pressure, T is the fluid temperature, q is the specific humidity of the fluid, ρ is the density, ν is the kinematic viscosity, α is the thermal diffusivity, and Γ is the diffusivity of water vapor. As presented, a numerical solution to Eqs. (1)–(4) reflects DNS since the material constants ν, α, and Γ are given as molecular quantities. Extension to LES is straightforward, in which case the field variables represent filtered quantities, and the diffusivities are replaced by a subfilter model based on Deardorff (1980). The details of this model are not provided here (see Moeng, 1984 for example); it is instead emphasized that the equations being solved can be easily switched to LES mode with no appreciable increase in cost.

At the same time, the code is equipped to integrate the trajectory of many individual Lagrangian point particles. The point particle approximation is commonly made in settings where the particles are smaller than the smallest scales of the turbulent flow (Balachandar, 2009), and is commonly used as a tool for studying dispersed phases in turbulence (Balachandar and Eaton, 2010). If in addition to being small the particle density ρ_p is much larger than the fluid velocity ρ, each particle's position, velocity, temperature, and radius can be described as follows (Maxey and Riley, 1983; Pruppacher and Klett, 1997):

$$\frac{d\mathbf{x_p}}{dt} = \mathbf{v_p}, \tag{5}$$

$$\frac{d\mathbf{v_p}}{dt} = \frac{1}{\tau_p}(\mathbf{u_f} - \mathbf{v_p}), \tag{6}$$

$$\frac{dT_p}{dt} = \frac{1}{\tau_T}\left(T_f - T_p\right) + \frac{\gamma_T}{r_p}\frac{dr_p}{dt}, \tag{7}$$

$$\frac{dr_p}{dt} = \gamma_q \frac{r_p}{\tau_p}\left(q_f - q_*\right). \tag{8}$$

Here, $\mathbf{v_p}$ refers to the velocity of a single particle, T_p is the particle temperature, and r_p is the particle radius. Several parameters appear in these expressions: τ_p is the inertial time constant of the particle and represents how quickly a particle can adjust to the local fluid velocity, τ_T is the thermal time constant of the particle and represents how quickly the particle can adjust its temperature through convective heat exchange, and γ_T and γ_q are coefficients which include quantities such as the fluid and particle specific heats, the particle density, the latent heat of the particle material, and the mass and heat transfer convection coefficients. Since the details of these expressions are not the focus of this work, and since the proposed GPU-accelerated algorithm is meant to apply to any point particle model under the broad structure of Eqs. (5)–(8) (e.g. Mashayek, 1998; Miller and Bellan, 1999; Russo et al., 2014), we refer the reader to Helgans and Richter (2016) and Peng and Richter (2017) for an explicit discussion of these terms in our particular setup. We also note that other forms of the particle momentum equation, including terms such as the added mass or lift forces (Maxey and Riley, 1983) for studying lighter-than-fluid particles (e.g. bubbles), can be easily added as well. In these situations, we would anticipate even larger performance enhancements via GPU acceleration, since these extra calculations, namely for the velocity derivatives and their interpolations, would be done on the GPU as well.

Furthermore, other forms of the particle momentum equations, including terms such as the added mass or lift forces (Maxey and Riley, 1983) for studying lighter-than-fluid particles (e.g. bubbles), are ideal candidates for GPU acceleration since their calculation can be entirely performed on the GPU device.

The linking of Eqs. (5)–(8) to the Eulerian fields of velocity, temperature, and humidity occurs through the $\mathbf{u_f}$, T_f, and q_f terms, which are Eulerian quantities interpolated to the particle location. The code is configured such that the user can choose between sixth-order Lagrange or trilinear interpolation. The particles therefore evolve primarily based on differences between their own velocity, temperature, and surface humidity (q_*) and the surrounding fluid. The particle temperature can also vary due to latent heat exchange via evaporation/condensation (second term on the right hand side of Eq. (7)).

Numerically, Eqs. (1)–(4) are solved using a pseudospectral discretization in the horizontal, periodic x and y directions, and second order finite differences in the wall-normal z direction. Time integration is performed for all particle and field quantities using a third-order Runge–Kutta (RK) scheme. Incompressibility of the fluid velocity field is enforced via solving a pressure Poisson equation at each RK stage, and the advection operators in Eqs. (3) and (4) are solved in skew-symmetric form. A typical snapshot of the fluid and particle solution is shown in Fig. 1, which shows instantaneous particle positions and streamwise velocity fluctuations for

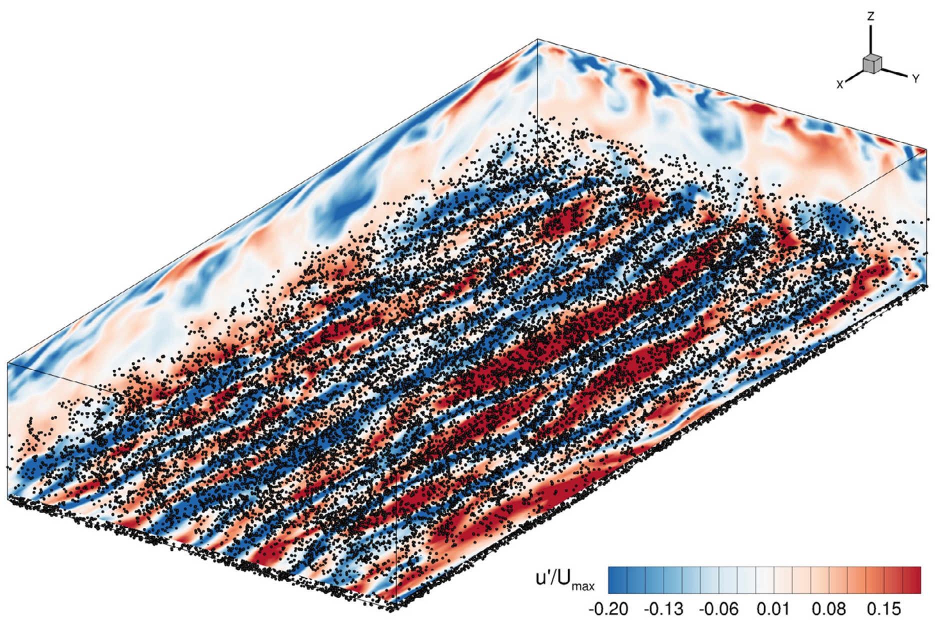

Fig. 1. Instantaneous snapshot of particle-laden turbulent channel flow simulated using the default configuration of the NTLP model. Particles in the lower half of the domain are shown as black dots, and colors represent normalized streamwise velocity fluctuations u'/U_{max}, where U_{max} is the maximum mean velocity at the channel centerline. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

a particle-laden turbulent channel flow configured according to the benchmark case of Marchioli et al. (2008).

The underlying CPU version of the code is written in Fortran and based on MPI for parallelization, and it has been shown to exhibit favorable scaling up to 16,384 processors across multiple architectures including Cray, SGI, and IBM (see e.g. Sullivan and Patton, 2011). In its default configuration, the Cartesian domain is decomposed over two dimensions, however the Lagrangian particles and Eulerian grid points are treated differently.

The Eulerian flow domain is decomposed over MPI processes across the y and z dimensions — see Fig. 2(a). With this decomposition, each processor contains the entire solution array in the x direction at each (y, z) point for ease in performing fast Fourier transformations (FFTs); MPI communication is required to perform FFTs in the y direction. Each processor contains a halo in the z direction for use in computing vertical derivatives.

The particles are instead decomposed over processors in the (x, y) plane — see Fig. 2(b). The reason for this difference in domain decomposition is that in wall-bounded turbulent flows, the dynamics of inertial particles is such that they tend to drift and accumulate near the top and/or bottom walls of the domain (Sardina et al., 2012). Therefore for purposes of load balancing across MPI processes, the particles must reside within columns aligned normal to the walls of the domain. As a result, MPI communication is required to “transpose” the field data distributed according to Fig. 2(a) into a format aligned with Fig. 2(b) so that interpolation can be performed on each Lagrangian particle. For instance in the schematic in Fig. 2, processor 0 must communicate with processors 2 and 4 in order to reconstruct the full velocity, temperature, and humidity fields required for interpolation by the particles which reside on processor 0. This nearly all-to-all communication step is relatively expensive, accounting for around 20% of the overall particle computation, and is replaced by transfer of data to the GPU device in the present implementation.

The overall structure of a given time step is that during each RK stage, the flow is updated via Eqs. (1)–(4), followed by the particle quantities via Eqs. (5)–(8). In previous studies we consider two-way coupling between the particles and the surrounding flow, but in the current setup we restrict our focus to cases where the particles do not feed back onto the flow.

The model configuration which we intend to improve with GPU acceleration is one where the Eulerian mesh is modest in size (on the order of 128^3 or 256^3 grid points), but where very large numbers of particles are desired. In this work we utilize a standard computational cluster and use a fixed set of 64 processors (Intel Xeon E5-2650 cores, Infiniband connectivity), focusing on the performance as the number of Lagrangian particles is increased. In its original configuration, the overall particle computation becomes more expensive than that of the flow computation around 10^6 particles.

2.2. GPU acceleration

In a wide range of applications, such as the numerical study of cloud droplets (Grabowski and Wang, 2012) or when using Lagrangian particles to represent reacting continuous phases (Benson and Bolster, 2016), very large numbers of particles may be desired while keeping the Eulerian grid fixed. Given the independent nature of the Lagrangian particles, especially in the limit of one-way coupling, the point-particle method aligns well with the advantages of GPU architecture since many individual threads can be operated upon simultaneously. Therefore to achieve our goal of simulating tens to hundreds of millions of particles transported by a turbulent flow, we recognize that under the circumstances described in the previous section, it is far more cost effective to exploit GPU parallelization rather than to simply increase CPU resources.

The basic strategy is therefore to offload the entire particle calculation to one or more GPUs, so that the particles themselves reside entirely on the GPU hardware (see Fig. 3). To do this, the Eulerian velocity, temperature, and humidity fields must be transferred at every RK stage, and therefore the GPU acceleration is deemed worthwhile only if it outweighs the cost of sending data to the device.

The present hardware configuration is such that four CUDA-based NVIDIA GPU devices are hosted by a single computational

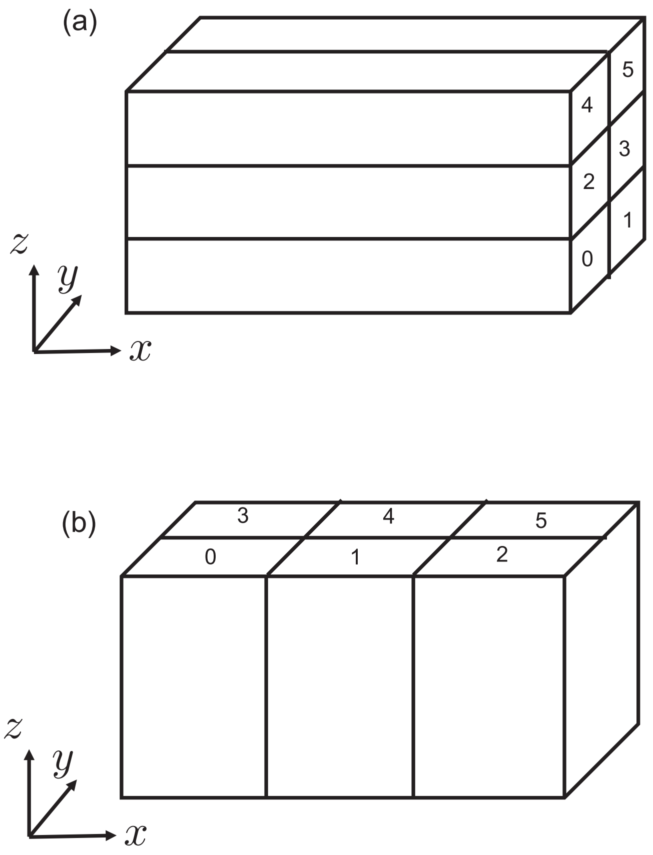

Fig. 2. (a) Schematic of the MPI domain decomposition for the Eulerian flow calculations. Example here shown for 6 MPI processes. (b) Schematic of the MPI domain decomposition for the Lagrangian particles, shown for the same 6 MPI processes.

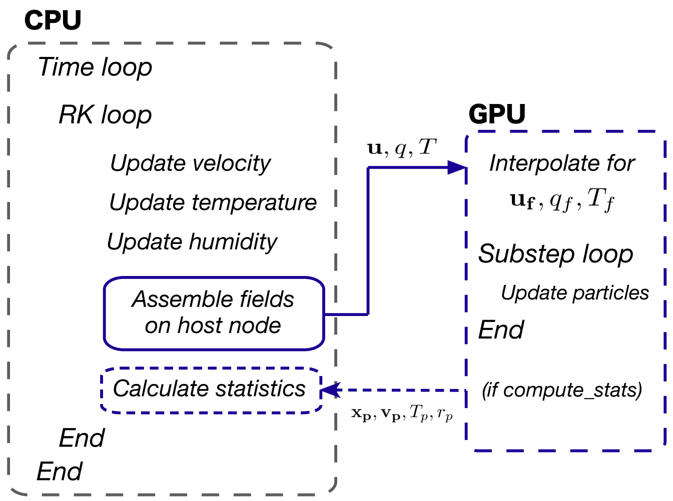

Fig. 3. Schematic of basic code structure. Within each RK loop, Eulerian fields are assembled and transferred, and while the GPU interpolates the local fluid properties and computes the trajectories of all particles, the CPU continues asynchronously. Particle information is retrieved from the GPU for calculation of statistics at specified times.

node. Once the flow variables have been updated at each RK stage, they are sent via MPI to the host node, which then initiates the transfer of the full velocity, temperature, and humidity fields to each of the GPU devices (solid arrow in Fig. 3). Note that for very large grid sizes, the current method may not be practical since a single node must reconstruct the full Eulerian fields at each RK stage. For the current hardware, this upper limit on grid size is around 256^3, since larger grids would both require more time for collection on the GPU host node (this scales nearly linearly with grid size), as well as take too much memory on the GPU device itself. A 512^3 flow field, for example, would take nearly half of the on-board GPU memory (approximately 5GB of the total 12GB), limiting the number of particles to only roughly 10^7. For grids 512^3 or larger, GPU acceleration would likely require each CPU node hosting its own GPU device, where each GPU only tracks particles which live in the domain subregion solved by that CPU node. Once a particle crosses from one region of the domain to another, GPU-to-GPU communication would be required, much like the MPI communication required in the original CPU configuration. This strategy is not discussed here.

Once the transfer of field data to GPU has been initiated by the host node, the CPU nodes continue the flow simulation asynchronously while the GPU devices update the particles. Particle data is only ever recovered from the GPU devices when dispersed phase statistics are computed, and this occurs at a user-specified frequency. At the onset of the simulation, the total requested number of particles is divided among the available GPU devices, and the total number of particles each device can track is limited only by on-board memory (in this case 12GB per device).

Within this algorithm, a few key choices can be made which factor into the balance between accuracy and performance:

1. Interpolation order for computing flow quantities at the particle locations
2. Precision of the Eulerian fields transferred to the GPU device
3. Sub-time-stepping the particle equations to resolve faster processes than the flow time step

These points will be highlighted in the following section.

3. Performance

For the overall computational algorithm outlined in the previous section, we present in this section the performance enhancements achieved by offloading particle calculations to the GPU devices. It is demonstrated that GPU acceleration allows for very large numbers of particles to be integrated with significant computational savings over the CPU cluster, while at the same time enabling multiresolution time integration that can resolve particle time scales which would otherwise be computationally expensive. For the following results, all calculations have been made on a cluster of Intel Xeon E5-2650 (Ivy Bridge) cores connected via Mellanox FDR non-blocking Infiniband. A specialized host node is included in the cluster which contains four NVIDIA Titan X Pascal GPUs, all of which can be used simultaneously. Each GPU contains 12GB of device memory, which results in a maximum of roughly 60 million particles that can be stored on each device in addition to the Eulerian fields. Unless otherwise specified, results are shown using 64 CPU cores and Eulerian grids of size 128^3.

Fig. 4 shows the overall performance for varying number of GPUs using trilinear interpolation and single-precision Eulerian fields. Here, several features are noteworthy. First, above a crossover of roughly 2×10^6 particles, computing trajectories on the GPU becomes significantly more cost effective. For 2.4×10^8 particles across four GPU devices, a speedup of approximately 14 is observed compared to the CPU calculation. Also shown in Fig. 4 is that the transfer time required for the GPU to retrieve the full Eulerian fields takes a constant time of roughly 0.06 s per time step; at low numbers of particles this transfer completely dominates the total GPU particle integration time — see Fig. 5, which provides the percent of a single time step taken by (1) the CPU to GPU transfer, (2) the flow calculation, and (3) the particle calculation on the GPU.

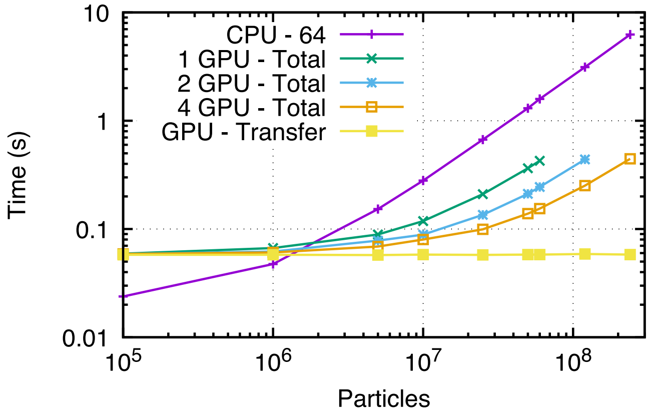

Fig. 4. Time taken per time step as a function of particle number, comparing 64 CPU cores to one, two, and four GPU devices. Also shown is the transfer time taken to send Eulerian field data to the GPU hardware, which is independent of particle number and number of GPU devices. Eulerian fields are transferred in single precision, and the flow grid has a size of 128^3.

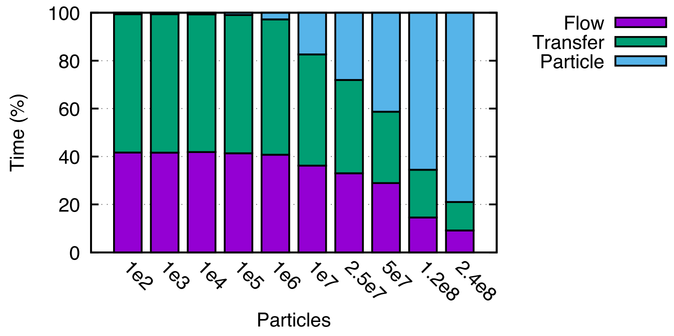

Fig. 5. Percentage breakdown of a single time step for the flow solver (CPU), particle solver (GPU), and transfer time, as a function of particle number.

Finally, Fig. 4 also shows that this algorithm scales nearly perfectly with GPU number. Since each GPU device calculates its own portion of the overall particles independently, and since the transfer time from the CPU host node to the GPU devices occurs simultaneously (i.e. not sequentially), going from one to four GPU devices reduces the particle integration time by GPU by a factor of four (excluding transfer time). Conversely, going from one to four GPU devices increases the capacity from roughly 60 million particles to 240 million particles with essentially zero overall increase in particle calculation time.

3.1. Precision of the Eulerian fields

The default configuration noted above consists of assembling and transferring the full Eulerian velocity, temperature, and humidity fields in single precision. Doing so reduces the MPI communication required to assemble the fields on the GPU host node, and reduces the transfer time from the host node to the GPU device.

Fig. 6 compares the performance between treating these fields as single versus double precision. The figure clearly shows that the total transfer time (blue and yellow lines) is indeed roughly doubled when going from single to double precision, and therefore makes the overall particle calculation on four GPU devices more expensive than those shown in Fig. 4. It also indicates that the change from single to double precision increases the crossover point where the GPU performs faster than the CPU — from roughly 2×10^6 particles to 7×10^6. While the advantage of using single precision fields is to reduce data transfer costs, the potential disadvantage is the loss of accuracy in the interpolated field quantities

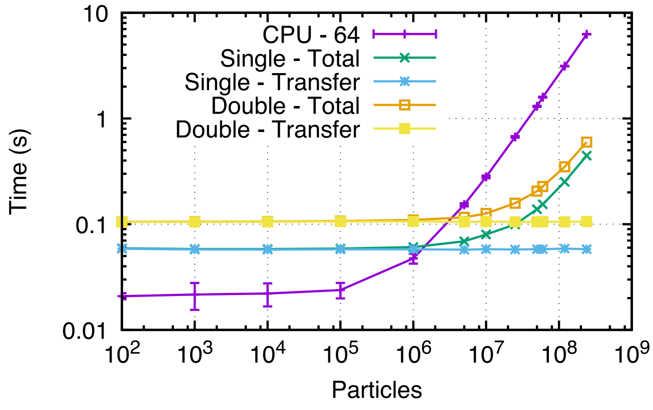

Fig. 6. Time taken per time step as a function of particle number, comparing cases where the Eulerian velocity, temperature, and humidity fields are represented in single versus double precision.

at the particle locations, and therefore in the accuracy of the dispersed phase statistics.

To validate this change from double to single precision, Fig. 7(a) and (b) compare the vertical profiles of mean particle streamwise velocity V_x^+ and streamwise root-mean-square (RMS) velocity $v_{x,rms}^+$ against four participants of the international benchmark case of Marchioli et al. (2008), who each use a slightly different numerical scheme. In this benchmark, turbulent pressure-driven channel flow laden with one-way coupled solid particles is computed for multiple particle sizes; see Fig. 1 for a snapshot of this flow. For brevity we report only the case for $St = 5$, where St is the particle Stokes number, a measure of its inertial response to the surrounding fluid.

The figures show that there is virtually no difference in the computed velocity statistics of the particles as a result of using single versus double precision Eulerian fields, and that both solutions lie directly among the benchmark solutions of Marchioli et al. (2008). We also note that the double precision GPU case is identical, within roundoff error, to a CPU solution with exactly the same configuration. Based on Fig. 7(a) and (b), we therefore recommend the use of single precision Eulerian fields to be transferred to the GPU hardware.

3.2. Interpolation order

In the original version of the code, a user specifies either sixth-order Lagrange interpolation or trilinear interpolation for obtaining flow data at the particle location (other interpolation schemes have been thoroughly evaluated and could similarly be implemented; see for example (Yeung and Pope, 1988) or Balachandar and Maxey (1989)). Since the entire Eulerian velocity, temperature, and humidity fields are transferred to the GPU device, the order of interpolation has no effect on the data transfer cost; it does, however, affect memory access by the GPU device, and sixth-order interpolation requires more code branching than the trilinear (to handle interpolation near walls).

As shown in Fig. 8, this results in a significant computational cost for higher-order interpolation schemes. Beyond 5×10^6 particles, the GPU outperforms the CPU regardless of interpolation order, but a substantial price is paid for switching from second to sixth order due to the random memory accesses required by a large interpolation stencil. At 240 million particles, the second order interpolation is approximately six times faster than the sixth order interpolation.

To assess the computational accuracy of switching the interpolation order, the benchmark case of Marchioli et al. (2008) is again used. Fig. 9(a) and (b) present the same particle statistics as in

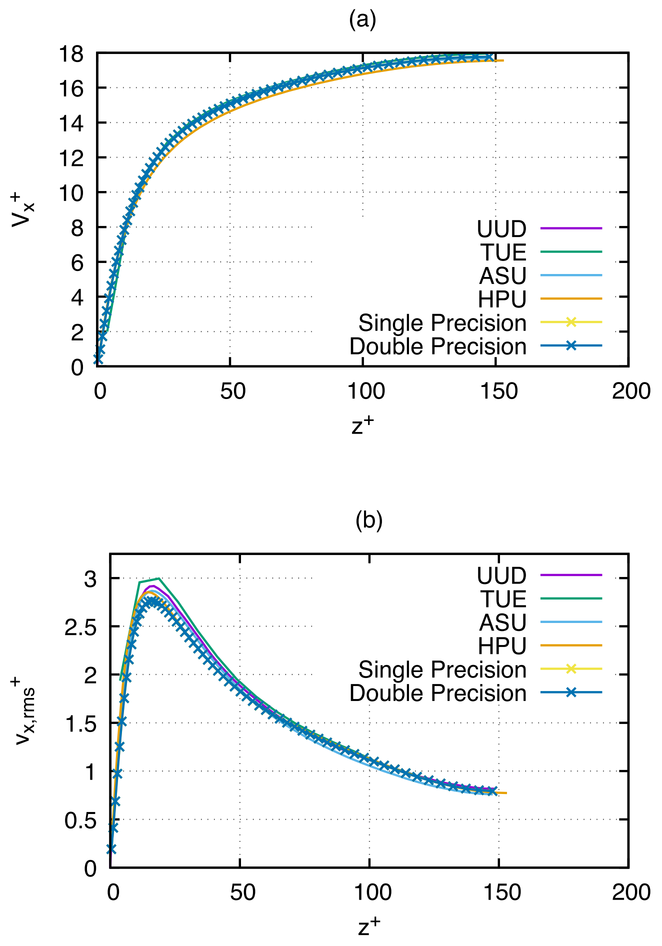

Fig. 7. For the benchmark particle-laden turbulent channel flow of Marchioli et al. (2008), a comparison of the GPU-based mean particle velocity against the four groups *UUD, TUE, ASU*, and *HPU*. The yellow line is the GPU calculation based on single precision Eulerian fields, and the blue line is the GPU calculation using double precision Eulerian fields. The superscripts "+" refer to normalizing the coordinate z and mean velocity V_x using turbulent wall units. See Marchioli et al. (2008) for more details. Figure (b) is the same as figure (a), except comparing streamwise RMS velocities $v_{x,\,rms}$ against the benchmark solutions of Marchioli et al. (2008). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

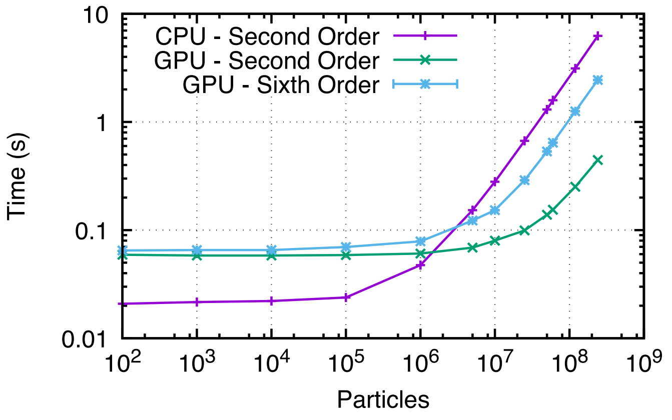

Fig. 8. Time taken per time step as a function of particle number, comparing cases using sixth-order Lagrange interpolation versus trilinear interpolation.

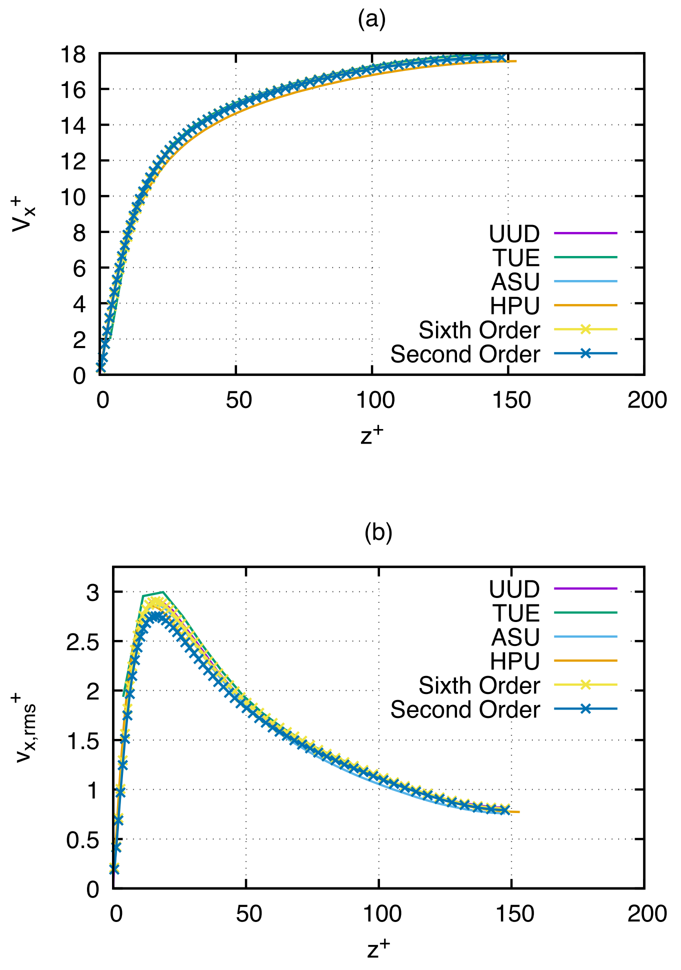

Fig. 9. Same as for Fig. 7, except now comparing the GPU-calculated particle statistics for sixth versus second order interpolation.

Fig. 7(a) and (b), and compare the effects of using second versus sixth order interpolation for flow properties.

In general, the decrease of interpolation order from sixth to second leaves the mean velocity statistics unchanged, while slightly reducing the peak of the RMS velocity which is located at $z^+ \approx 25$. The deviation observed from switching between sixth and second order interpolation is within 5%. Depending on the specific application, this filtering of second order particle statistics may be deemed unacceptable, but trilinear interpolation is used frequently in the particle-laden turbulence literature (Bernardini, 2014; Park et al., 2017) and we therefore use it for our baseline GPU configuration, given its significant computational savings.

3.3. Multiresolution time stepping

The use of the GPU device for the particle integration allows for the straightforward implementation of multiresolution time stepping, where the particles take a specified number of substeps for each flow time step. As denoted in Fig. 3, once the Eulerian field data is transferred to the GPU device, a time integration loop based on the same RK3 scheme as the outer loop is performed on the particle equations while keeping the interpolated values frozen. This strategy removes the need for additional interpolation calculations, and allows for particle integration at time steps lower than the flow requires. This occurs when the timescales τ_p or τ_T of Eqs. (6) and (7) are small relative to the time step required by the DNS.

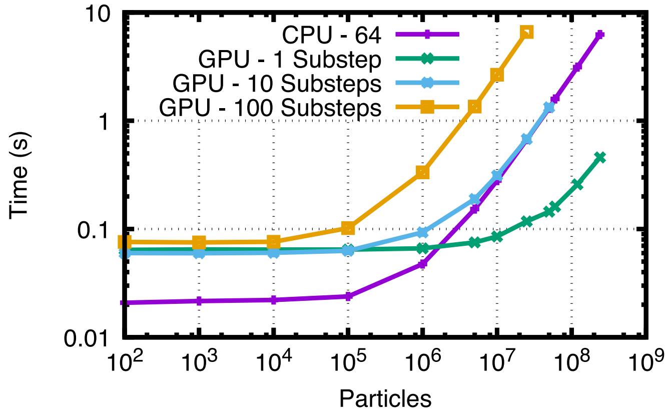

Fig. 10. Time taken per timestep as a function of particle number, comparing the original CPU version of the code with the GPU accelerated code with varying number of sub-integration timesteps for particle dynamics.

Fig. 10 again shows the time taken per flow time step as a function of particle number, now comparing the CPU-based calculation to the GPU accelerated code with varying numbers of particle integration substeps. It is shown that at high numbers of particles, the GPU code can compute 10 substeps for each particle for the same original cost as a single time step on a cluster of 64 CPUs. Since at high particle numbers the GPU time is dominated by particle calculations (see Fig. 5), increasing the number of substeps from 10 to 100 increases the cost by slightly less than a factor of 10.

While Fig. 10 does indeed show that the GPU acceleration can provide substepping essentially for free compared to the original CPU implementation, it can be somewhat better demonstrated using a specific example. Here, a slightly modified version of the benchmark flow of Marchioli et al. (2008) is simulated where the particle timescales τ_p and τ_T are significantly smaller than the time step of the flow. Particle evaporation is also turned on, using the model described in detail in Helgans and Richter (2016), and we use 10^6 particles. The top and bottom boundaries are set to the same temperature $T_0 = 300.15$ K, and the relative humidities of the top and bottom wall are held fixed at 95% and 100%, respectively.

Instead of using a particle Stokes number of 5 as in the above comparisons to Marchioli et al. (2008), we reduce the particle size so that the dimensionless Stokes number is $St = 0.24$. For a Courant-Friedrichs-Lewy (CFL) number of 0.4, the ratio of the nominal flow time step to the particle acceleration timescale τ_p would then be $\Delta t/\tau_p = 0.55$ — too large to resolve the particle accelerations accurately. This simply means that the time step Δt will in this case be limited by particle processes (via τ_p) instead of the CFL condition.

To demonstrate the advantage of the GPU, we perform two simulations: one simulation using the CPU code with a time step 5 times smaller than suggested by the CFL condition (so that $\Delta t/\tau_p = 0.1$), and one simulation using the same original time step associated with a CFL number of 0.4, but using 10 substeps on the GPU.

Fig. 11 presents the mean particle radius (Fig. 11(a)) and particle temperature (Fig. 11(b)) for this flow as a function of channel height as computed by these two simulations, and shows that the simulations produce nearly identical results. Of note, however, is that the CPU version of the code took 36 h of wall time to complete the simulation, while the GPU version of the code took 5 h.

We close this section by further highlighting the flow physics which can be uncovered by utilizing GPU acceleration, specifically the multiresolution substepping. In particle-laden turbulent channel flows, turbophoresis is one of the key processes which governs mean particle distributions, and for inertial particles this leads to high concentrations near the walls (Sardina et al., 2012). For evaporating droplets, this processes still occurs, but changes in droplet sizes can cause asymmetric distributions of particles, since turbophoretic drift will be altered by the growth or reduction in droplet mass due to evaporation and condensation. This effect is described in detail in Russo et al. (2014) for instance, and in the present case we can extend this to very small particles, whose size can change very rapidly, thus modifying their inertial response to the turbulence.

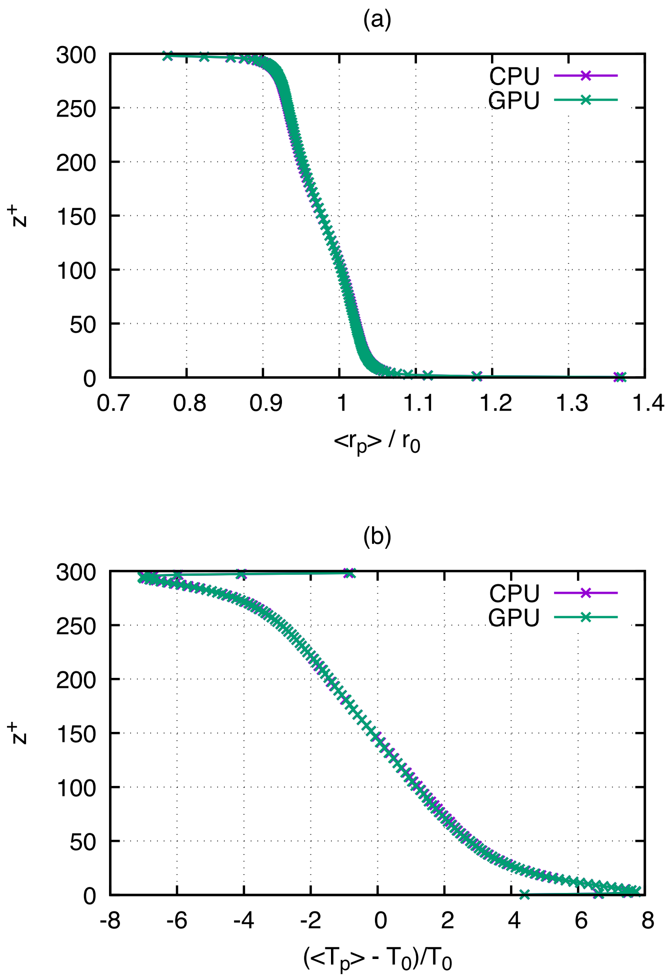

Fig. 11. (a) Mean particle radius $\langle r_p \rangle$ normalized by initial radius r_0 as a function of normalized channel height z^+. (b) Mean particle temperature difference $(< T_p > -T_0)/T_0$, where T_0 is the temperature of the top and bottom boundary. *CPU* refers to the original code simulating turbulent channel flow with evaporating droplets, using a timestep small enough to resolve fast particle response times. *GPU* refers to using the timestep provided by a CFL condition but using 10 substeps to resolve fast particle evolution.

For the same turbulent channel flow used in Fig. 11 (i.e., top and bottom walls held fixed at T_0 and the bottom and top relative humidities at 100% and 95%, respectively), Fig. 12(a) presents the normalized particle number concentration near the bottom wall for three different particle Stokes numbers: $St = [0.06, 0.24, 0.96]$. For the Stokes numbers considered, a maximum near-wall number concentration is seen for particles of $St \approx 1$.

From Fig. 11, it is clear that at steady-state, particles near the top (relatively dry) wall are generally reduced in size, while those near the bottom (saturated) wall are larger in size. Due to the consequent differences in turbophoresis, this leads to a change in the near-wall number concentration when evaporation is turned off, even for the very small particle size of $St = 0.06$. This is shown

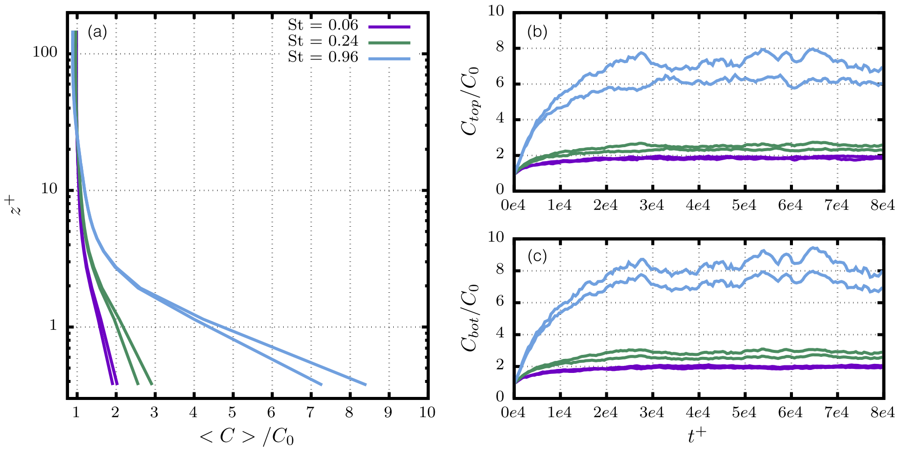

Fig. 12. Particle number concentration statistics for evaporating droplets in the same channel as in Fig. 11. Solid lines denote evaporating droplets, while dashed lines denote nonevaporating particles of the same initial size. Line colors refer to the Stokes numbers given in the legend. (a) Normalized number concentration near the bottom wall, where C_0 is the bulk number concentration in the channel, (b) time evolution of normalized particle number concentration at the top wall for the three Stokes numbers, and (c) time evolution of particle number concentration at the bottom wall. C_{bot} and C_{top} refer to the horizontally averaged concentration in the grid node nearest the bottom and top walls, respectively.

in Fig. 12 as the difference between the solid (evaporating) and dashed (nonevaporating) lines for each particle size. Fig. 12(a) shows the steady-state near-wall concentration for each evaporating/nonevaporating pair, while Fig. 12(b) and (c) show the time evolution of the normalized near-wall concentration at both the top and bottom walls. In the latter two figures, it is clear that while larger (here $St = 0.96$) particles take longer to approach steady near-wall concentrations, condensation leads to a surplus of particles at the bottom wall while leading to a deficit of particles near the top wall. Without evaporation, the average number of particles at both walls are the same.

4. Conclusions

This work demonstrates the advantages of using GPUs to accelerate simulations of particle-laden turbulent flows by integrating the particle equations of motion entirely on one or more NVIDIA GPU devices. This method is based on the CPU sending the entire Eulerian fields of velocity, temperature, and humidity to the GPU, and asynchronously continuing its calculations while the GPU integrates the trajectories of hundreds of millions of particles.

This method is designed to enhance simulations where modest grid sizes (up to, say, 256^3) are used but while many particles (more than 10^6) are desired. The use of multiple GPU devices scales nearly perfectly, since each computes the trajectories of its portion of independent, one-way coupled particles. Speedup of up to 14x is found at particle numbers above 10^7.

We consider three factors representing the tradeoff between performance and accuracy. The use of single precision Eulerian fields being transferred to the GPU device reduces the transfer time by a factor of two, while showing essentially zero change in typical particle statistics. The order of interpolation from the Eulerian grid to the particle location significantly changes the computational time due to memory access and code branching. While this does modify second order statistics somewhat, for many applications this is likely an acceptable compromise. Finally, the use of GPU acceleration can be used to efficiently integrate sub-timesteps of the particle equation, in order to resolve fast particle dynamics which would otherwise require a smaller flow time step. It is shown that for roughly 10 sub-steps, the overall cost for large numbers of particles is essentially equal to the original CPU calculation.

As noted above, this algorithm requires Eulerian grids that can be assembled and transferred in their entirety to the GPU device. It is this step which is the major bottleneck for further performance enhancements. One future strategy to consider is to distribute GPUs to each computational node, but this adds the complexity of now having to transfer particles from one GPU device to another. Thus, the current setup is ideal for systems of modest flow size but large particle numbers, and extension to two-way coupling (feedback onto the flow field) is relatively straightforward. Further extension to four-way coupling (particle-particle collisions) is conceptually straightforward as well, but only when utilizing a single GPU device; since particle collisions and neighbor searches are typically expensive, however, this restriction to a single GPU device may still be worthwhile.

Acknowledgements

This work was supported by the Office of Naval Research (ONR) under Grant No. N00014-16-1-2472. Computational resources were provided by the Notre Dame Center for Research Computing.

References

Aissa, M., Verstraete, T., Vuik, C., 2017. Toward a GPU-aware comparison of explicit and implicit CFD simulations on structured meshes. Comput. Math. Appl. 74, 201–217.
Balachandar, S., 2009. A scaling analysis for pointparticle approaches to turbulent multiphase flows. Int. J. Multiphase Flow 35 (9), 801–810.
Balachandar, S., 2010. Turbulent dispersed multiphase flow. Annu. Rev. Fluid Mech. 42, 111–133.
Balachandar, S., Maxey, M.R., 1989. Methods for evaluating fluid velocities in spectral simulations of turbulence. J. Comput. Phys. 83, 96–125.
Benson, D.A., Bolster, D., 2016. Arbitrarily complex chemical reactions on particles. Water Resour. Res. 52, 9190–9200.
Bernardini, M., 2014. Reynolds number scaling of inertial particle statistics in turbulent channel flows. J. Fluid Mech. 758, R1.
Deardorff, J.W., 1980. Stratocumulus-capped mixed layers derived from a three-dimensional model. Boundary Layer Meteorol. 18 (4).
Gomez-Gesteira, M., Crespo, A.J.C., Rogers, B.D., Dalrymple, R.A., Dominguez, J.M., Barreiro, A., 2012. SPHYsics - development of a free-surface fluid solver - part 2: efficiency and test cases. Comput. Geosci. 48, 300–307.
Gonzalez, C., Richter, D.H., Bolster, D., Bateman, S., Calantoni, J., Escauriaza, C., 2017. Characterization of bedload intermittency near the threshold of motion using a lagrangian sediment transport model. Environ. Fluid Mech. 17, 111–137.
Grabowski, W.W., Wang, L.-P., 2012. Growth of cloud droplets in a turbulent environment. Annu. Rev. Fluid Mech. 45, 293–324.
van Heerwaarden, C.C., van Stratum, B.J.H., Heus, T., Gibbs, J.A., Fedorovich, E., Mellado, J.-P., 2017. MicroHH 1.0: a computational fluid dynamics code for direct numerical simulation and large-eddy simulation of atmospheric boundary layer flows. Geosci. Model Dev. Discuss. 96, 1–33.

Helgans, B., Richter, D.H., 2016. Turbulent latent and sensible heat flux in the presence of evaporative droplets. Int. J. Multiphase Flow 78, 1–11.
Hérault, A., 2010. SPH on GPU with CUDA. J. Hydraul. Res. 48, 74–79.
Khajeh-Saeed, A., Perot, J.B., 2013. Direct numerical simulation of turbulence using GPU accelerated supercomputers. J. Comput. Phys. 235, 241–257.
Manuel, R.L., Bull, J., Crabill, J., Romero, J., Sheshadri, A., Ii, J.E.W., Williams, D., Palacios, F., Jameson, A., 2014. Verification and validation of hifiLES: a high-order LES unstructured solver on multi-GPU platforms. In: 32nd AIAA Applied Aerodynamics Conference, Atlanta, GA, pp. 1–27.
Marchioli, C., Soldati, A., Kuerten, J.G.M., Arcen, B., Taniere, A., Goldensoph, G., Squires, K.D., Cargnelutti, M.F., Portela, L.M., 2008. Statistics of particle dispersion in direct numerical simulations of wall-bounded turbulence: results of an international collaborative benchmark test. Int. J. Multiphase Flow 34, 879–893.
Mashayek, F., 1998. Direct numerical simulations of evaporating droplet dispersion in forced low mach number turbulence. Int. J. Heat Mass Transf. 41 (17), 2601–2617.
Maxey, M.R., Riley, J.J., 1983. Equation of motion for a small rigid sphere in nonuniform flow. Phys. Fluids 26 (4), 883–889.
Michalakes, J., Vachharajani, M., 2008. Gpu acceleration of numerical weather prediction. Parallel Process. Lett. 18 (04), 531–548.
Mielikainen, J., Huang, B., Huang, H.L.A., Goldberg, M.D., 2012. GPU Acceleration of the updated goddard shortwave radiation scheme in the weather research and forecasting (WRF) model. IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. 5 (2), 555–562.
Miller, R.S., Bellan, J., 1999. Direct numerical simulation of a confined three-dimensional gas mixing layer with one evaporating hydrocarbon-droplet-laden stream. J. Fluid Mech. 384, 293–338.
Minden, V., Smith, B., Knepley, M.G., 2013. Preliminary implementation of PETSc using GPUs. In: Yuen, D.A., Wang, L., Chi, X., Johnsson, L., Ge, W., Shi, Y. (Eds.), GPU Solutions to Multi-scale Problems in Science and Engineering. Springer Berlin Heidelberg, pp. 131–140.
Moeng, C.-H., 1984. A large-eddy-simulation model for the study of planetary boundary-layer turbulence. J. Atmos. Sci. 41 (13), 2052–2062.
Mokos, A., Rogers, B.D., Stansby, P.K., 2017. A multi-phase particle shifting algorithm for SPH simulations of violent hydrodynamics with a large number of particles. J. Hydraul. Res. 55, 143–162.
Obrecht, C., Kuznik, F., Tourancheau, B., Roux, J.J., 2013. Multi-GPU implementation of the lattice boltzmann method. Comput. Math. Appl. 65, 252–261.
Park, G.I., Bassenne, M., Urzay, J., Moin, P., 2017. A simple dynamic subgrid-scale model for LES of particle-laden turbulence. Phys. Rev. Fluids 2 (4), 044301.
Peng, T., Richter, D.H., 2017. Influence of evaporating droplets in the turbulent marine atmospheric boundary layer. Boundary-Layer Meteorology. In Press.
Pruppacher, H.R., Klett, J.D., 1997. Microphysics of Clouds and Precipitation, 2nd Kluwer Academic Publishers.
Richter, D.H., Garcia, O., Astephen, C., 2016. Particle stresses in dilute polydisperse, two-way coupled turbulent flows. Phys. Rev. E 93, 013111.
Richter, D.H., Sullivan, P.P., 2013. Momentum transfer in a turbulent particle-laden Couette flow. Phys. Fluids 25, 053304.
Richter, D.H., Sullivan, P.P., 2014. The sea spray contribution to sensible heat flux. J. Atmos. Sci. 71 (2), 640–654.
Russo, E., Kuerten, J.G.M., van der Geld, C.W.M., Geurts, B.J., 2014. Water droplet condensation and evaporation in turbulent channel flow. J. Fluid Mech. 749, 666–700.
Salvadore, F., Bernardini, M., Botti, M., 2013. GPU Accelerated flow solver for direct numerical simulation of turbulent flows. J. Comput. Phys. 235, 129–142.
Sardina, G., Schlatter, P., Brandt, L., Picano, F., Casciola, C.M., 2012. Wall accumulation and spatial localization in particle-laden wall flows. J. Fluid Mech. 699, 50–78.
Schalkwijk, J., Griffith, E.J., Post, F.H., Jonker, H.J.J., 2012. High-performance simulations of turbulent clouds on a desktop PC: exploiting the GPU. Bull. Am. Meteorol. Soc. 93 (3), 307–314.
Schalkwijk, J., Jonker, H.J.J., Siebesma, A.P., Van Meijgaard, E., 2015. Weather forecasting using GPU-based large-eddy simulations. Bull. Am. Meteorol. Soc. 96, 715–723.
Sullivan, P.P., McWilliams, J.C., Moeng, C.-H., 1994. A subgrid-scale model for large-eddy simulation of planetary boundary-layer flows. Boundary Layer Meteorol. 71, 247–276.
Sullivan, P.P., Patton, E.G., 2011. The effect of mesh resolution on convective boundary layer statistics and structures generated by large-eddy simulation. J. Atmos. Sci. 68, 2395–2415.
Tomov, S., Nath, R., Ltaief, H., Dongarra, J., 2010. Dense linear algebra solvers for multicore with gpu accelerators. In: 2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), pp. 1–8.
Winkler, D., Meister, M., Rezavand, M., Rauch, W., 2017. GpuSPHASE - a shared memory caching implementation for 2d SPH using CUDA. Comput. Phys. Commun. 213, 165–180.
Yeung, P.K., Pope, S.B., 1988. An algorithm for tracking fluid particles in numerical simulations of homogeneous turbulence. J. Comput. Phys. 79, 373–416.