The present manuscript introduces a GPU-enabled solver, LCS.jl, for simulating inertial particle dynamics in turbulent flows. The solver is a Julia port of an existing CPU-only Fortran code. Both use MPI for distributed computing on CPUs and GPUs. The main objective of the paper is that one can write state-of-the-art solvers using a modern programming language like Julia, which enables fast prototyping and portability across a variety of CPU and GPU computing platforms in a single codebase and using generic code. The Julia port matches the performance of the previous Fortran implementation on CPUs while enabling massive speed-ups on recent GPU platforms, where it displays very good scalability. The paper also discusses the chosen strategies (mainly regarding ghost cell and particle communication between MPI processes) which enable good GPU performance. Other than that, the models and numerical methods presented in the paper are standard and seem to be the same as in the Fortran implementation. The LCS.jl solver is open-source and might benefit the geophysical community in some way, although unfortunately that is not very clearly stated in the paper.

General comments

I have a few reservations regarding the current manuscript version. As a first general comment, the paper needs more context on what this solver may be used for. It seems to target the simulation of cloud microphysics, and in particular the dynamics of droplets in clouds, but that is not discussed much. It would be interesting to know under which regimes does the model apply, considering that the particle phase is treated using an idealised point-particle approach which neglects several terms of the original Maxey-Riley-Gatignol equations (which should be cited) and that a one-way coupling approximation is used (fluid $\longrightarrow$ particles). In particular, is this approximation relevant when such a large number of particles is simulated? Or stated differently, what is the point of simulating so many particles (of the order of the number of Eulerian grid points)? Is this standard practice in this kind of simulation?

Specific comments

1. Line 39: the difficulties associated with particle communication (I guess between GPUs and between different compute nodes) arrive somewhat abruptly, since domain decomposition has not been discussed yet. It would make sense to briefly introduce the usual domain decomposition strategies used in MPI-based simulations on structured grids.

2. Line 42: related to the above remark, it is not clear what "Conventional CPU-based implementations handle this communication sequentially" means. In cases where 1 CPU core = 1 MPI rank (= 1 subdomain), then the implementation is obviously sequential. But if a different configuration is used (say 1 MPI rank = 48 CPU cores), then I do not see why the implementation could not be parallel. In particular, there is no reason the "prefix-scan" strategy described later in the paper could not be applied to such a pure-CPU configuration.

3. Line 50: the sentence "However, …" is redundant with the sentence after it ("To the authors' knowledge, …") and should be removed.

4. Line 53: if the Oceananigans.jl code (Ramadhan et al. 2020) is limited to tracer particles, then I wouldn't call it a *multiphase* DNS solver (and I think their developers would agree with this).

5. Line 54: why would particle communication costs be more significant for inertial particles? This is not at all clear and deserves some explanation. I guess this might be the case if the particle distribution is spatially inhomogeneous due to preferential concentration / clustering? Or is it because one needs to transfer more information?

6. Line 66: as already pointed out by another referee, equations (1) and (2) should be given a name (incompressible Navier–Stokes).

7. For completeness, describe the forcing term in some more detail (at a minimum, give a mathematical expression for it), even if it was already introduced in a previous paper. It seems like the forcing depends on the instantaneous velocity field? How does it compare to schemes generally used in pseudo-spectral DNS? In particular, is it the same forcing proposed by Lamorgese, Caughey & Pope (Phys. Fluids, 2005)?

8. One may wonder whether there is any reason to use finite-difference methods instead of the more accurate pseudo-spectral Fourier methods, since this solver seems to be specifically written for periodic domains. The evaluation of the forcing would also be much simpler (and likely more efficient) in a Fourier-based solver. Any comment on this?

9. Line 80: to enforce the incompressibility condition, iterations of the HSMAC scheme are performed "until the RMS of the velocity divergence falls below δ/Δ" with δ set to 10^{-3} and with Δ the grid spacing. There seems to be an issue in the definition of this convergence criterion, as the expression $\boldsymbol{\nabla} \cdot \boldsymbol{u} < \delta/\Delta$ is not dimensionally consistent. Namely, $\boldsymbol{\nabla} \cdot \boldsymbol{u}$ has the units of an inverse time (T^{-1}) while δ/Δ is homogeneous to an inverse length (L^{-1}), assuming δ is a nondimensional coefficient as the paper seems to imply. Any explanation on this? Am I missing something?

10. Line 91: discuss the equations used for the particle phase. The authors should at the very least explain that they use a point-particle approximation and one-way coupling (fluid $\longrightarrow$ particles). How do these equations compare to the classical Maxey-Riley-Gatignol equations? What are the physical hypotheses entering equations (5)-(6)? (heavy and small particles, no added mass, no history effects, ...). In actual simulations, are these hypotheses really met (in particular regarding the small particle size compared to the Kolmogorov scale)?

11. Line 101: trilinear interpolation can be very inaccurate for particle tracking in numerical simulations. Have the authors compared with higher order interpolations (for example Hermite)?

12. Line 107: AcceleratedKernels.jl, a relatively recent package built on top of KernelAbstractions.jl, provides very similar functionality to the proposed `Parallel.foraxes` (with e.g. `AcceleratedKernels.foreachindex`). Would it make sense to switch to AcceleratedKernels.jl?

13. Line 109: "An explicit data movement design was also adopted." It is not clear what this means, does this refer to explicit D2H and H2D transfers of data? Isn't this the only way when using KernelAbstractions.jl or the GPUArrays.jl interface?

14. In Listing 1, it is not clear whether the indices `2:N-1` are subdomain indices or indices associated to the global domain. This should be clarified.

15. Line 131: if I understood correctly, the "time-blocking" strategy freezes ghost (halo) cells during a certain number of iterations. Doesn't this lead to some numerical error, especially near the subdomain boundaries? Has this error been evaluated in the present implementation?

16. Line 139: the prefix-scan strategy for particle communication should also work on CPUs. Have you tried this, and does this lead to any gain compared to the sequential implementation? Alternatively, have you tried using atomic operations to perform this step?

17. Line 168: show definition of Re_λ and of the Taylor scale λ.

18. Line 170: why does the energy spectrum definition include a $4\pi k^2/N_k$ factor? I would imagine this factor to be close to 1 for large k. Is it some kind of correction allowing to obtain a smoother energy spectrum at small k?

19. Table I: how is the Reynolds number tuned, is it via the forcing term only (viscosity seems to be kept constant)? Do you impose some forcing amplitude or perhaps a target energy injection rate? This control parameter should be included in the table for reproducibility.
20. Table I: show the numerical values of characteristic scales: integral scale, Taylor scale and Kolmogorov scale.
21. Line 172: "an inertial range spanning two decades". In fact, from Fig. 1, it seems like the inertial range spans only about one decade (roughly from $kl_\eta = 2 \cdot 10^{-2}$ to $2 \cdot 10^{-1}$) in the largest simulations.
22. Line 175: what is the definition of the radial distribution function g? What do r and R refer to? (In Sec. 2.2, the particle radius was denoted r; is it still the case here or did the notation change?)
23. Figure 2: what are the particle sizes relative to the Kolmogorov scale (r/η) for all the simulations? Is the point-particle approximation valid in all cases?
24. Figure 2: what is the "reference model" appearing in the legend (not mentioned nor cited in the caption or the main text). How do you explain the differences with the reference DNS, especially at high inertia (large Stokes)? How do you interpret the nonmonotonous behaviour with St?
25. Table 2: just to be sure, each node has $4 \times 48 = 192$ CPU cores? (so $2 \times 192 = 384$ threads if one uses hyperthreading, which I'm not sure is beneficial for this kind of code)
26. Figure 3: how do you explain that time blocking alone (red curve) doesn't lead to any noticeable speedup?
27. Table 3: as pointed out above, would it be possible to implement and compare with a "CPU-delegated parallel" case, perhaps based on a CPU version of the "prefix-scan" strategy?
28. Line 227: "LCS.jl uses identical kernel code for both CPU and GPU". Would it make sense to implement CPU-specific kernels with the same kind of optimisation as in the Fortran version? It would be easy to dispatch based on the KernelAbstractions backend to these kernels. It would be interesting to see if this closes the gap in performance with the Fortran version of the code.
29. Starting from line 244: it is not clear what the authors call "1 CPU". The authors write "1 CPU (96 threads)", so this would correspond to half a socket according to Table 2 (i.e. $1/4$ of a node), but this is contradicted later in the paper. I do not think this should be called a CPU. More generally, I would avoid talking about number of CPUs since it is confusing (give number of sockets or cores instead).
30. The authors seem to be using hyperthreading, but this is generally not recommended for this kind of simulation. Have you tried disabling hyperthreading?
31. Line 248: the discussion on optimisation efficiency is not very clear. If I understood correctly, the ideal is to use 4 threads = 1 MPI process = 1 subdomain? And what if you disable hyperthreading?
32. Figure 5: it would be interesting to add more points to this figure to better see the scalings.
33. Table 4: the "CPUs" column seems to imply that 1 node = 2 "CPUs", which is opposite to what seems to be implied earlier (1 node = 4 "CPUs"). Instead of listing the number of "CPUs" (which, again, is confusing), perhaps list the number of MPI processes and the number of CPU cores per MPI process.
34. Table 5: on the last column, show the actual percentage (72%) to be consistent with the caption and avoid confusion.

35. Line 279: “Since each node has 4 GPUs and 2 CPUs”. Again, what is a “CPU”? Here it seems to be a full socket.
36. Line 287: “particle statistics computation runs on a GPU”. What kind of particle statistics are computed? Is it costly? Is it done at each timestep?