Response to Referee 1

Manuscript ID: egusphere-2026-2214
Title: *LCS.jl v1.0: A High-Performance, Multi-Platform Computational Model in Julia for Turbulent Particle-Laden Flows*

All page and line numbers below refer to the revised manuscript.

General note on the performance measurements

During the revision, we found that the performance measurements reported in the discussion manuscript had been obtained with an earlier version of the code that still contained an unnecessary merge-sort operation. This operation was removed later during the prefix-scan analysis, but the correction had not been reflected in those earlier performance measurements. The operation does not affect the physical results, but its removal results in a measurable improvement in computational performance. We therefore repeated the performance measurements so that the revised results consistently correspond to the corrected implementation.

The repeated measurements showed an improvement in absolute performance of approximately 20–30% for representative GPU cases. Because the improvement is not uniform across resource counts, some scaling efficiencies also changed. In particular, the strong-scaling efficiency at the largest GPU scale changed from the previously reported value of approximately 85% to approximately 70%, while the weak-scaling efficiency at the largest GPU scale is approximately 80%. We have therefore revised the corresponding performance claims, figures, tables, and discussion throughout the manuscript.

For the repeated scaling measurements, we also adopted a consistent measurement and convergence procedure. The reported time per time-integration step is the maximum wall-clock time over all MPI processes, corresponding to the parallel critical path. Warm-up steps are excluded from the statistics. CPU and single-GPU benchmarks used 100 warm-up steps, whereas multi-GPU benchmarks on TSUBAME4.0 used 1000 warm-up steps unless otherwise noted. Measurement stability was assessed from the relative difference between the first and second halves of the measurement window and from the temporal trend, with 3% used as the convergence criterion.

All performance values and quantities derived from them in the revised manuscript have been updated using these repeated measurements.

Responses to individual comments

We thank the referee for the careful review and constructive comments. We have addressed each comment below and revised the manuscript accordingly.

Comment 1

> Please check the manuscript for typos and language issues. For example, in the abstract, “model implemented in Julia language” should be “model implemented in the Julia language”.

Response: We agree with the suggested correction.

Changes in manuscript (p. 1, line 5): We have corrected the phrase in the abstract and performed a general language revision throughout the manuscript.

Comment 2

> Please use well-known names of equations, e.g., incompressible Navier-Stokes equations when introducing (1)-(2).

Response: Equations (1) and (2) are the incompressible Navier-Stokes equations.

Changes in manuscript (p. 3, line 72): We have identified Eqs. (1) and (2) explicitly as the incompressible Navier-Stokes equations.

Comment 3

> Is the density ρ assumed to be constant in (2)?

Response: The fluid density is assumed to be constant.

Changes in manuscript (p. 3, line 75): We have stated explicitly, when introducing the governing equations, that the fluid density is constant.

Comment 4

> Line 78: There are many two-stage, second-order Runge–Kutta methods. Please specify which one is used in the manuscript.

Response: The explicit midpoint method is used for both the fluid and particle equations. It uses a half-step first stage followed by a full-step update.

Changes in manuscript (p. 4, lines 92–94; p. 5, lines 127–128): We have specified explicitly that the explicit midpoint method is used for both the fluid and particle equations.

Comment 5

> Page 3: Please provide a (brief) discussion of why you chose the numerical discretizations as you did.

Response: The numerical methods used in LCS.jl were intentionally inherited from the original LCS framework so that the Julia implementation retains the established numerical formulation of the original model. These methods were originally selected with large-scale parallel computation in mind. The finite-difference formulation relies primarily on local stencil operations and therefore requires only neighboring-subdomain communication for the spatial discretization. The conservative formulation of the advection term also preserves the appropriate discrete conservation properties. In addition, the Reduced Communication Forcing method retains large-scale spectral forcing while reducing the global communication required for its evaluation. Onishi et al. (2011) compared the finite-difference/RCF formulation with a spectral formulation and showed comparable turbulence statistics while reducing the communication required for parallel computation.

Changes in manuscript (p. 4, lines 105–112): We have added a brief rationale for the numerical discretizations and cited the relevant previous work, including Onishi et al. (2011).

Comment 6

> Please provide concrete references for the parameter values you chose, e.g., the kinematic viscosity ν in line 70 and the particle response time τ_p etc. in lines 95ff.

Response: The parameter values and particle-response-time formulation follow the original LCS formulation of Onishi et al. (2015). In that study, the kinematic viscosity is $\nu = 1.5 \times 10^{-5}$ m^2/s and the particle-to-fluid density ratio is $\frac{\rho_p}{\rho} = 8.43 \times 10^2$ for air and water at 1013 hPa and 298 K. The particle response time follows the same formulation. We have cited Onishi et al. (2015) explicitly where these parameters are introduced. The reference to Rowe and Henwood (1961) for the nonlinear drag correction has been retained.

Changes in manuscript (p. 3, lines 75–76; pp. 4–5, lines 117–123): We have cited Onishi et al. (2015) where the physical parameters and particle response time are introduced, while retaining the reference to Rowe and Henwood (1961) for the nonlinear drag correction.

Comment 7

> Sections 3.2.2 and 3.2.3: Julia code is compiled just ahead of execution. Do the performance comparisons include compilation time? What is the overhead of compilation (likely parallel, performed on each rank instead of just once in traditional compiled languages)? The last question is particularly interesting for Section 3.2.4, in particular for increasing number of MPI ranks.

Response: The reported steady-state performance measurements exclude JIT compilation by discarding the warm-up interval. We also quantified the first-step overhead separately. We define T_{first} as the wall-clock time of the first complete time-integration step and T_{prod} as the converged production time per step, and estimate the one-time overhead as

$$T_{\text{first}} - T_{\text{prod}}.$$

This timing does not include process creation, MPI launcher startup, or other initialization performed before the time-integration measurement begins. The measured overhead was approximately 4.6–5.8 s for the multi-process CPU cases and 12.0–13.4 s for the GPU strong-scaling cases. For the GPU cases, the critical-path overhead remained approximately constant from 8 to 256 GPUs rather than increasing in proportion to the number of MPI processes.

Changes in manuscript (p. 9, lines 235–246; p. 11, lines 315–319): We have clarified that steady-state timings exclude JIT compilation and added the first-step overhead definition and measured overhead separately from the steady-state performance results.

Comment 8

> Please also provide some scaling results from serial to a full node and the baseline used for the scaling results shown in the manuscript.

Response: We performed an additional fixed-size intra-node CPU strong-scaling measurement using 1, 2, 4, 8, 16, 32, 64, 96, 128, and 192 physical CPU cores. All reported points satisfied the convergence criterion. From one to 192 physical CPU cores, the wall time decreased from 29.14 s to 0.4222 s, corresponding to a speedup of $\frac{29.14}{0.4222} = 69.0$ for one complete node.

Changes in manuscript (p. 11, lines 320–328; p. 12, lines 329–346): We have added the intra-node CPU strong-scaling results and stated explicitly the baseline resource count used for each scaling series.

Comment 9

> Please clarify early whether single or double precision is used in the simulations. I assume it is double precision (because of line 195), but it would be good to state this concretely.

Response: All simulations and performance measurements in this study were performed in double precision.

Changes in manuscript (p. 3, line 70): We have stated near the beginning of the Methods section that all computations reported in this study use double-precision floating-point arithmetic.

Comment 10

> Since nearly all the results use the same kind of hardware and performance portability is emphasized strongly already in the abstract, please provide some performance results on different architectures (AMD GPUs etc.).

Response: We performed an additional full LCS.jl time-integration benchmark on two GPU architectures using the CUDA and AMDGPU backends. For grid sizes $N = 128, 192, 256$, and 384, with $N_\mathrm{p} = \left(\frac{N}{2}\right)^3$ particles, the AMDGPU-to-CUDA execution-time ratios were 1.21, 1.34, 1.48, and 1.54, respectively. Thus, the measured performance on the two GPU platforms differs by approximately a factor of 1.2–1.5 and remains of the same order. The comparison is intended as a portability test rather than a detailed vendor-to-vendor hardware comparison.

Changes in manuscript (pp. 12–13, lines 355–360; p. 23, Table 5): We have added the full time-integration-step CUDA/AMDGPU benchmark as an additional portability test on a different GPU architecture and clarified the grid and particle counts used in the benchmark.

Comment 11

> It is great that the code has been published under an open-source license. However, the repository contains just a single commit. Thus, it is unclear whether the community will benefit from (likely to be anticipated) further developments. Moreover, the code documentation should be improved. For example, the manuscript highlights Parallel.foraxes in Listing 1, which cannot be found when searching the documentation of the repository.

Response: The single visible commit results from releasing the public repository as a squashed snapshot; it does not indicate that development has stopped. Development of LCS.jl is continuing.

Changes in manuscript (p. 14, lines 407–410; repository documentation): We have expanded the repository documentation to describe the configuration workflow, backend selection, and `Parallel.foraxes`. In the manuscript, the Code availability section now distinguishes the current development version from the exact archived publication version on Zenodo.

Response to Referee 2

Manuscript ID: egusphere-2026-2214
Title: *LCS.jl v1.0: A High-Performance, Multi-Platform Computational Model in Julia for Turbulent Particle-Laden Flows*

All page and line numbers below refer to the revised manuscript.

General note on the performance measurements

During the revision, we found that the performance measurements reported in the discussion manuscript had been obtained with an earlier version of the code that still contained an unnecessary merge-sort operation. This operation was removed later during the prefix-scan analysis, but the correction had not been reflected in those earlier performance measurements. The operation does not affect the physical results, but its removal results in a measurable improvement in computational performance. We therefore repeated the performance measurements so that the revised results consistently correspond to the corrected implementation.

The repeated measurements showed an improvement in absolute performance of approximately 20–30% for representative GPU cases. Because the improvement is not uniform across resource counts, some scaling efficiencies also changed. In particular, the strong-scaling efficiency at the largest GPU scale changed from the previously reported value of approximately 85% to approximately 70%, while the weak-scaling efficiency at the largest GPU scale is approximately 80%. We have therefore revised the corresponding performance claims, figures, tables, and discussion throughout the manuscript.

For the repeated scaling measurements, we also adopted a consistent measurement and convergence procedure. The reported time per time-integration step is the maximum wall-clock time over all MPI processes, corresponding to the parallel critical path. Warm-up steps are excluded from the statistics. CPU and single-GPU benchmarks used 100 warm-up steps, whereas multi-GPU benchmarks on TSUBAME4.0 used 1000 warm-up steps unless otherwise noted. Measurement stability was assessed from the relative difference between the first and second halves of the measurement window and from the temporal trend, with 3% used as the convergence criterion.

All performance values and quantities derived from them in the revised manuscript have been updated using these repeated measurements.

Responses to individual comments

We thank the referee for the careful and constructive review. The comments helped us clarify the physical assumptions, numerical methods, and performance evaluation of LCS.jl. We have addressed each comment below and revised the manuscript accordingly.

General comment

> 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 → 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?

Response: LCS.jl is a Julia reimplementation of the Lagrangian Cloud Simulator framework developed in our previous studies. The original LCS was developed to provide reference data for cloud-microphysics studies by individually tracking droplets, while the present work focuses on the fundamental turbulent-flow and inertial-particle components and on their portable high-performance implementation. The particle model used in the present study is intended for dilute suspensions of small, heavy inertial particles. It uses a one-way-coupled point-particle approximation and retains the drag force with a finite-particle-Reynolds-number correction. Added-mass and history forces, gravity, particle feedback on the carrier flow, collisions, and hydrodynamic interactions between particles are not included in the present simulations. The validity of one-way coupling is determined by physical quantities such as particle volume fraction and mass loading, rather than by the total number of tracked computational particles alone. In the original LCS applications, the particle mass and volume fractions were sufficiently dilute that flow modulation could be neglected. A large number of particles is required to obtain statistically converged particle statistics, particularly pair statistics such as the radial distribution function. In the performance benchmarks, large particle numbers also provide a sufficiently demanding workload for evaluating the performance of the particle solver.

Changes in manuscript (p. 1, lines 17–22; p. 5, lines 129–137): We have expanded the Introduction and particle-model description to clarify the intended application context, the point-particle and one-way-coupling assumptions, their relation to the Maxey-Riley-Gatignol framework, and the reason for using large particle ensembles.

Comment 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.

Response: LCS.jl uses a three-dimensional Cartesian domain decomposition in which each MPI process owns one subdomain. Neighboring processes exchange halo data for the Eulerian flow solver, while particles crossing a subdomain boundary are transferred to the corresponding neighboring process.

Changes in manuscript (p. 2, lines 40–42): We have introduced the three-dimensional Cartesian domain decomposition before discussing particle communication and explained the Eulerian halo exchange and particle migration between neighboring MPI processes.

Comment 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.

Response: The previous wording was too general. The sequential behavior referred specifically to the original LCS implementation, in which storage positions for migrating particles were determined using a sequential counter-based packing procedure. This is an implementation property rather than

an inherent limitation of CPU execution. The prefix-scan formulation used in LCS.jl can also be executed on the CPU backend.

Changes in manuscript (p. 2, lines 47–50; p. 10, lines 277–280): We have revised the text to distinguish the sequential packing algorithm used in the original LCS implementation from CPU execution in general and stated that the prefix-scan formulation can also run on the CPU backend.

Comment 3

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

Response: The sentence is redundant and can be removed without changing the content.

Changes in manuscript (pp. 2–3, lines 56–60): We have removed the redundant sentence.

Comment 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).

Response: Oceananigans.jl should not be described as an inertial-particle multiphase DNS solver in this context.

Changes in manuscript (pp. 2–3, lines 56–60): We have described Oceananigans.jl as a Julia-based flow solver supporting Lagrangian tracer-particle tracking, rather than as an inertial-particle multiphase DNS solver.

Comment 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?

Response: Our intended point was that inertial particles carry additional dynamical state compared with tracer particles. In addition to particle position, particle velocity must be migrated and packed when a particle crosses a subdomain boundary. In addition, unlike the statically defined halo communication of the Eulerian flow solver, particle communication is dynamic because the number and destinations of migrating particles change during the simulation.

Changes in manuscript (p. 2, lines 43–47): We have explained that inertial-particle communication transfers additional dynamical state and that the number and destinations of migrating particles vary dynamically during the simulation.

Comment 6

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

Response: Equations (1) and (2) are the incompressible Navier-Stokes equations.

Changes in manuscript (p. 3, line 72): We have identified Eqs. (1) and (2) explicitly as the incompressible Navier-Stokes equations.

Comment 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)?

Response: RCF depends on the instantaneous velocity field but is not the same forcing scheme as that used by Lamorgese et al. (2005). The velocity field is first coarse-grained before the Fourier transform, and the amplitudes of the forced low-wavenumber modes are rescaled to maintain their prescribed energy. This reduces the globally communicated data compared with applying the Fourier transform to the full-resolution velocity field. Onishi et al. (2011) provide the detailed formulation.

Changes in manuscript (p. 3, lines 77–87): We have expanded the existing RCF description, provided its mathematical formulation, clarified its distinction from the forcing of Lamorgese et al. (2005), and referred to Onishi et al. (2011) for the detailed formulation.

Comment 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?

Response: The principal reason for retaining the finite-difference formulation is its suitability for massively parallel computation. The present formulation was inherited from the original LCS framework, in which most spatial operations use local finite-difference stencils and therefore require only neighboring-subdomain halo communication. A pseudo-spectral formulation provides high spatial accuracy but requires global Fourier transforms, whose communication cost becomes increasingly important in large distributed-memory calculations. The RCF method was developed specifically to retain large-scale spectral forcing while reducing this global communication in a finite-difference solver. We therefore retained the finite-difference formulation in LCS.jl to preserve the numerical framework of the original LCS and its parallel scalability.

Changes in manuscript (p. 4, lines 105–112): We have added the rationale for retaining the finite-difference formulation and cited the previous finite-difference/RCF comparisons with spectral calculations.

Comment 9

> Line 80: to enforce the incompressibility condition, iterations of the HSMAC scheme are performed "until the RMS of the velocity divergence falls below $\frac{\delta}{\Delta}$" 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 $\nabla \cdot \boldsymbol{u} < \frac{\delta}{\Delta}$ is not dimensionally consistent. Namely, $\nabla \cdot \boldsymbol{u}$ has the units of an inverse time (T^{-1}) while $\frac{\delta}{\Delta}$ 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?

Response: The convergence criterion in the original manuscript was expressed using nondimensional solver variables, but this was not stated explicitly. We have clarified the nondimensionalization and rewritten the criterion accordingly in the revised manuscript. The tolerance $\delta = 10^{-3}$ is dimensionless.

Changes in manuscript (p. 4, lines 95–98): We have explicitly defined the nondimensional variables used by the HSMAC solver and rewritten the convergence criterion consistently in nondimensional form, with $\delta = 10^{-3}$ as a dimensionless tolerance.

Comment 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 → 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)?

Response: The present simulations consider small, heavy particles using a one-way-coupled point-particle approximation. Drag is retained with a finite-particle-Reynolds-number correction. Added-mass and history forces, gravity, particle feedback on the carrier flow, collisions, and hydrodynamic interactions between particles are not included in the present simulations. For the validation cases, the particle radius and diameter satisfy

$$\frac{a}{\eta} \leq 0.21$$

and

$$\frac{d}{\eta} \leq 0.42,$$

respectively. Thus, the particle diameter remains below the Kolmogorov length scale for all validation cases, supporting the point-particle approximation.

Changes in manuscript (p. 5, lines 129–135; p. 8, lines 223–224): We have expanded the particle-model description, related it explicitly to the Maxey-Riley-Gatignol framework, and reported the particle-size ratios $\frac{a}{\eta} \leq 0.21$ and $\frac{d}{\eta} \leq 0.42$.

Comment 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)?

Response: We did not perform an additional interpolation-order comparison in the present study. Trilinear interpolation was inherited from the original LCS formulation of Onishi et al. (2015). The interpolation choice in the original framework was made with reference to previous comparisons including cubic Hermitian, cubic Lagrangian, and fifth-order Lagrangian interpolation, as discussed by Sundaram and Collins (1996).

Changes in manuscript (p. 5, lines 124–128): We have clarified that trilinear interpolation is inherited from the original LCS formulation and added the references to Onishi et al. (2015) and the higher-order interpolation comparisons discussed by Sundaram and Collins (1996).

Comment 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?

Response: AcceleratedKernels.jl indeed provides functionality that overlaps with the general parallel-iteration part of `Parallel.foraxes`. `Parallel.foraxes`, however, is a small internal abstraction that also incorporates iteration patterns specific to LCS.jl. In particular, it provides the red-black coloring variants used by the HSMAC solver. It also allows backend-specific specialization: for the CUDA backend, LCS.jl uses a dedicated implementation based directly on multidimensional CUDA thread indices, avoiding part of the generic indexing overhead. Thus, ordinary parallel loops could in principle be expressed using AcceleratedKernels.jl, whereas replacing the current abstraction as a whole would require retaining these solver- and backend-specific specializations.

Changes in manuscript (p. 5, lines 140–146): We have clarified the role of `Parallel.foraxes`, including the LCS-specific coloring patterns and backend specialization, and avoided implying that general backend-independent iteration is unique to LCS.jl.

Comment 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?

Response: The intended comparison was with directive-based programming models such as OpenACC. In LCS.jl, device-resident arrays are handled directly, and host-device transfers are explicitly invoked in the program where they are required, rather than relying on compiler-managed offloading and data movement. We did not intend to imply that explicit data transfer is unique to KernelAbstractions.jl or GPUArrays.jl.

Changes in manuscript (p. 5, lines 146–149): We have revised the text to explain that the intended comparison is with directive-based offloading approaches and clarified what is meant by explicit data movement in LCS.jl.

Comment 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.

Response: The indices `2:N-1` in Listing 1 refer to local subdomain indices, not global-domain indices.

Changes in manuscript (p. 6, Listing 1): We have stated explicitly that `2:N-1` denotes local subdomain indices and that N is the local subdomain size.

Comment 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?

Response: The time-blocking implementation does not freeze halo values while applying an approximate update. A wider halo is exchanged before the grouped HSMAC iterations, and the active computational domain is then successively contracted so that each stencil evaluation uses valid data. The method therefore preserves the stencil dependencies of the original HSMAC iterations and changes only the communication schedule; it does not introduce an additional numerical approximation at subdomain boundaries.

Changes in manuscript (p. 6, lines 170–175): We have clarified that time blocking uses a wider halo and successively contracts the active computational domain, preserving the stencil dependencies without freezing stale halo values.

Comment 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?

Response: The original LCS implementation used a sequential counter-based procedure to determine storage positions when packing migrating particles. The prefix-scan formulation used in LCS.jl is backend independent and can also be executed on CPUs. Atomic-based packing provides another possible parallel strategy. These alternatives were not compared in the present study.

Changes in manuscript (p. 10, lines 277–280): We have clarified that the prefix-scan formulation can also run on CPUs and noted atomic-based packing as another possible parallel strategy that was not evaluated in the present study.

Comment 17

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

Response: The Taylor-microscale Reynolds number and Taylor scale are defined as

$$\mathrm{Re}_\lambda = u_{\mathrm{rms}} \frac{\lambda}{\nu}$$

and

$$\lambda = \sqrt{15\nu \frac{u_{\mathrm{rms}}^2}{\varepsilon}},$$

respectively.

Changes in manuscript (p. 7, lines 208–210): We have added the definitions

$$\mathrm{Re}_\lambda = u_{\mathrm{rms}} \frac{\lambda}{\nu}$$

and

$$\lambda = \sqrt{15\nu \frac{u_{\mathrm{rms}}^2}{\varepsilon}}.$$

Comment 18

> Line 170: why does the energy spectrum definition include a $4\pi \frac{k^2}{N_k}$ 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?

Response: The factor $4\pi \frac{k^2}{N_k}$ compensates for the discrete number N_k of Fourier modes contained in each spherical shell. The spectrum is first averaged over the discrete modes in the shell, and the factor converts this average into an estimate corresponding to the continuum spherical-shell measure $4\pi k^2$.

Changes in manuscript (p. 8, lines 213–216): We have clarified that $4\pi\frac{k^2}{N_k}$ is a discrete mode-count correction converting the shell average to the continuum spherical-shell measure, rather than a smoothing operation.

Comment 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.

Response: The kinematic viscosity and reference length L_0 are fixed, and the bulk Reynolds number $\mathrm{Re} = U_0\frac{L_0}{\nu}$ is varied through the reference velocity U_0. RCF is configured through the target energy of the $k = 1$ shell; the values were adjusted slightly so that $\frac{u_{\mathrm{rms}}}{U_0} \approx 1$. For reproducibility, the revised Table 1 reports both Re and the configured nondimensional target energy $\frac{E_{1,\mathrm{target}}}{U_0^2}$ for every validation case.

Changes in manuscript (p. 7, lines 205–208; p. 8, Table 1): We have added Re and $\frac{E_{1,\mathrm{target}}}{U_0^2}$ to Table 1 and clarified how the validation conditions are controlled with fixed viscosity and reference length.

Comment 20

> Table I: show the numerical values of characteristic scales: integral scale, Taylor scale and Kolmogorov scale.

Response: These characteristic scales are available for each statistically stationary validation case and can be reported directly.

Changes in manuscript (p. 8, Table 1): We have added the numerical integral length scale, Taylor microscale, and Kolmogorov length scale for each case to Table I.

Comment 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.

Response: The referee is correct that the inertial range shown in Fig. 1 spans approximately one decade rather than two.

Changes in manuscript (p. 8, lines 217–218): We have revised "two decades" to "approximately one decade."

Comment 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?)

Response: For the monodisperse cases considered here, $R = 2a$, where a is the particle radius.

Changes in manuscript (p. 8, lines 220–223): We have defined the RDF $g(r)$ and clarified the meanings of the pair separation r and the contact separation R; for the monodisperse cases, $R = 2a$.

Comment 23

> Figure 2: what are the particle sizes relative to the Kolmogorov scale ($\frac{r}{\eta}$) for all the simulations? Is the point-particle approximation valid in all cases?

Response: For all validation cases, the particle radius satisfies $\frac{a}{\eta} \leq 0.21$, corresponding to a particle diameter of $\frac{d}{\eta} \leq 0.42$. The particle diameter therefore remains below the Kolmogorov length scale for all validation cases, supporting the point-particle approximation.

Changes in manuscript (p. 8, lines 223–224): We have reported $\frac{a}{\eta} \leq 0.21$ and $\frac{d}{\eta} \leq 0.42$ explicitly in the discussion of the validation cases.

Comment 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?

Response: The reference model is the RDF parameterization of Onishi and Seifert (2016). The remaining differences at larger St arise from the empirical parameterization of the RDF and are discussed in that study. The non-monotonic dependence on St reflects the change in the dominant clustering mechanism with particle inertia. For small St, particles are preferentially expelled from vortical regions and accumulate in high-strain regions. Clustering becomes strong when the particle response time becomes comparable to the small-scale turbulent timescale, i.e., for $\mathrm{St} = O(1)$. For larger St, particle motion becomes less responsive to the dissipative scales and clustering is increasingly governed by larger turbulent scales, leading eventually to weaker clustering.

Changes in manuscript (p. 8, lines 225–231; p. 18, Fig. 2): We have identified and cited the reference RDF parameterization of Onishi and Seifert (2016) in the figure caption and main text, noted that the differences at larger St reflect the empirical parameterization, and added a brief physical explanation of the non-monotonic St dependence.

Comment 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)

Response: Each TSUBAME4.0 node contains two AMD EPYC 9654 CPU sockets, with 96 physical CPU cores per socket and 192 physical CPU cores per node. Simultaneous multithreading was not used in the reported measurements.

Changes in manuscript (p. 11, lines 307–309; p. 17, Table 2): We have corrected the hardware description and consistently distinguished CPU sockets, physical CPU cores, MPI processes, Julia threads, and GPUs.

Comment 26

> Figure 3: how do you explain that time blocking alone (red curve) doesn't lead to any noticeable speedup?

Response: We repeated these measurements using the corrected code and the common measurement protocol. In the repeated measurements, time blocking alone produces a modest speedup of approximately 1.6–6.6%, depending on the GPU count. Time blocking reduces the number of halo exchanges, but this reduction is partly offset by the additional computation performed in the

enlarged halo region. This explains why the gain from time blocking alone remains modest. The combination of time blocking and communication-computation overlap was the fastest configuration for all tested cases from 8 to 64 GPUs, providing an improvement of approximately 5.6–10% over the unoptimized case.

Changes in manuscript (p. 9, lines 248–254; p. 19, Fig. 3): We have updated Fig. 3 and the associated discussion using the repeated measurements, including the modest gain from time blocking alone and the larger gain from combining time blocking with communication-computation overlap.

Comment 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?

Response: A CPU-delegated parallel packing implementation is possible, and the prefix-scan formulation itself can also be executed on a CPU. We did not add this additional case because Table 3 is intended to evaluate the cost of delegating processing of GPU-resident particle data to the CPU, rather than to compare optimized CPU and GPU packing kernels. Parallelizing CPU packing and unpacking could reduce those components, but the full-array D2H and H2D transfers would remain in a CPU-delegated implementation. The GPU-side prefix-scan implementation instead performs the packing while the particle data remain on the GPU.

Changes in manuscript (p. 10, lines 277–280): We have clarified that Table 3 evaluates the cost of CPU delegation for GPU-resident particle data and that a separate optimized CPU-parallel packing comparison is outside the scope of this benchmark.

Comment 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.

Response: CPU-specific kernels could further improve CPU performance. The primary design goal of the present implementation, however, is to retain a shared kernel implementation across CPU and GPU backends and to evaluate the performance portability obtained with that design. The current abstraction permits backend-specific specialization where useful. We have clarified the resulting trade-off between architecture-specific peak-performance tuning and maintaining a shared CPU/GPU kernel implementation.

Changes in manuscript (pp. 10–11, lines 290–297): We have added a discussion of the trade-off between CPU-specific optimization and maintaining shared CPU/GPU kernel code, and noted that further CPU-specific tuning may improve CPU performance at the cost of additional architecture-specific implementation complexity.

Comment 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).

Response: The terminology in the original performance section is ambiguous because "CPU" is used for different hardware/resource levels.

Changes in manuscript (pp. 11–12, lines 307–354): We have replaced ambiguous uses of "CPU" throughout the performance section with explicit terms such as CPU socket, physical CPU core, MPI process, Julia thread, and GPU.

Comment 30

> The authors seem to be using hyperthreading, but this is generally not recommended for this kind of simulation. Have you tried disabling hyperthreading?

Response: Simultaneous multithreading was not used in the reported CPU measurements. In the process/thread benchmark, one Julia thread was assigned to each physical CPU core.

Changes in manuscript (p. 11, lines 307–309; lines 320–322): We have stated explicitly that simultaneous multithreading was not used and that one Julia thread was assigned to each physical CPU core in the process/thread benchmark.

Comment 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?

Response: In this benchmark, each MPI process owns one subdomain, and the total number of physical CPU cores was fixed at 384. One Julia thread was assigned to each physical core, with simultaneous multithreading disabled. Among the tested configurations, four Julia threads per process gave the best performance, corresponding to a 2.18 times speedup relative to the 96-thread-per-process configuration. We do not imply that four threads per process is universally optimal.

Changes in manuscript (p. 11, lines 307–314): We have clarified the relationship among MPI processes, subdomains, Julia threads, and physical CPU cores, stated that simultaneous multithreading was disabled, and reported the best-performing process/thread balance among the tested configurations.

Comment 32

> Figure 5: it would be interesting to add more points to this figure to better see the scalings.

Response: We performed an additional fixed-size intra-node CPU strong-scaling benchmark using 1, 2, 4, 8, 16, 32, 64, 96, 128, and 192 MPI processes on one node. All reported points satisfied the convergence criterion. The wall time decreased from 29.14 s with one MPI process to 0.4222 s with 192 MPI processes, corresponding to a speedup of $\frac{29.14}{0.4222} = 69.0$.

Changes in manuscript (p. 11, lines 320–325; p. 20, Fig. 5): We have added the fixed-size intra-node strong-scaling series with 1, 2, 4, 8, 16, 32, 64, 96, 128, and 192 MPI processes on one node.

Comment 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.

Response: We agree that the original "CPUs" terminology was ambiguous. We revised the resource descriptions to distinguish CPU sockets, physical CPU cores, MPI processes, Julia threads, and GPUs explicitly. In Table 4, the former "CPUs" column is now labeled "CPU Sockets" to identify the hardware resource unambiguously.

Changes in manuscript (p. 12, lines 348–352; p. 20, Table 4): We have replaced the ambiguous "CPUs" terminology in Table 4 with "CPU Sockets" and used explicit resource terminology consistently throughout the performance section.

Comment 34

> Table 5: on the last column, show the actual percentage (72%) to be consistent with the caption and avoid confusion.

Response: The repeated CPU-only and heterogeneous measurements gave 71.0 s and 22.7 s per step, respectively, corresponding to an approximately 68% reduction in wall-clock time, or a speedup of approximately 3.13.

Changes in manuscript (p. 13, lines 371–374; p. 23, Table 6): We have reported the heterogeneous-execution improvement as a percentage and updated the value to approximately 68% using the repeated measurements.

Comment 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.

Response: Here, "CPU" refers to a CPU socket; the original terminology is therefore ambiguous.

Changes in manuscript (pp. 11–12, lines 327–328 and 348–349): We have replaced the expression with "four GPUs and two CPU sockets per node" and used the same terminology consistently.

Comment 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?

Response: The particle statistics computed in this benchmark are the radial distribution function (RDF) and the particle-number-density spectrum. Both are evaluated at every time step. The RDF requires particle-pair searching, binning, and accumulation. The particle-number-density spectrum is evaluated from the particle positions over multiple wavenumber shells and sampling directions and requires accumulation over the full particle population followed by MPI reduction. These statistics therefore constitute a substantial computational workload for the large particle population used in the benchmark.

Changes in manuscript (p. 13, lines 362–367): We have specified that the radial distribution function and particle-number-density spectrum are evaluated at every time step and briefly described the computational work required by each statistic.