Response Letter to Reviewer #1

High-Performance Geodynamics on GPUs Using the PETSc's CUDA Backend

[Ms. Ref. No.: egusphere-2026-2528]

[Kyeong-Min Lee[1], Deok-Kyu Jang[3], Cedric Thieulot[4], Byung-Dal So[1,2]*]

Text in ~~red strike-through~~: deleted from the manuscript.
Text in blue: added or revised.
Text in *italic*: comments from Reviewer #1.

The manuscript presents GPU-accelerated solvers for Stokes flow in geodynamic modelling. Matrices are assembled on the CPU, while PETSc's GPU backend and HYPRE perform the solver operations on the device (PCG on the Schur complement with inner GMRES+HYPRE).
The study is interesting and novel, provides a detailed comparison across many examples, and is strengthened by its emphasis on reproducibility. Despite some major concerns, the study is worthy of publication in my opinion:

Authors' reply:

We sincerely thank the reviewer for the careful evaluation of our manuscript and constructive comments, particularly regarding solver configuration, performance evaluation, and reproducibility. In the revised manuscript, we have addressed these comments through additional analyses, clarifications, and corresponding revisions, as detailed below.

Main point 1:
Too many and often repetitive results. The manuscript can be shortened and provide the same information and conclusions:

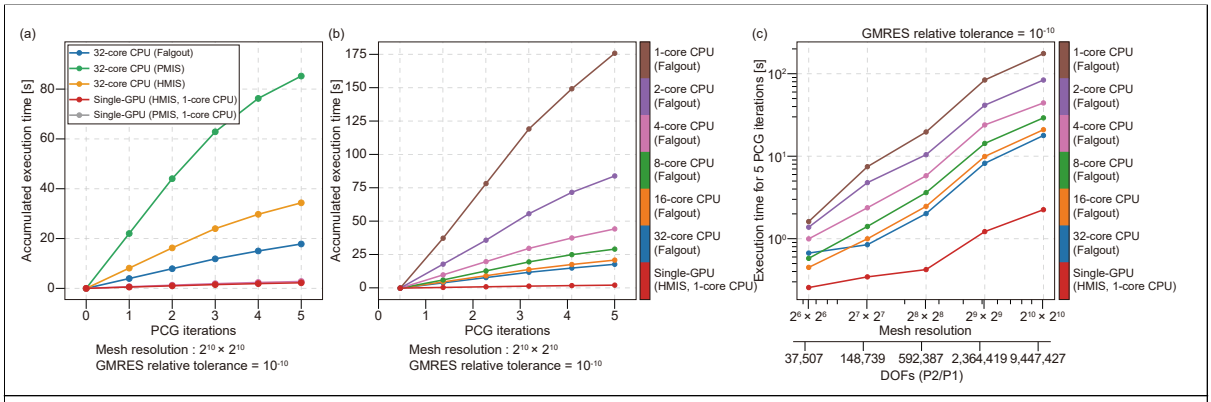
Authors' reply:

We appreciate the reviewer's helpful comment. We streamlined the Results section by reducing the number of main-text figures from 14 to 12 and moving the 3-D Rayleigh–Taylor example, including the original Figure 6c, to the Supplementary Material. Detailed changes addressing the three sub-comments are provided below.

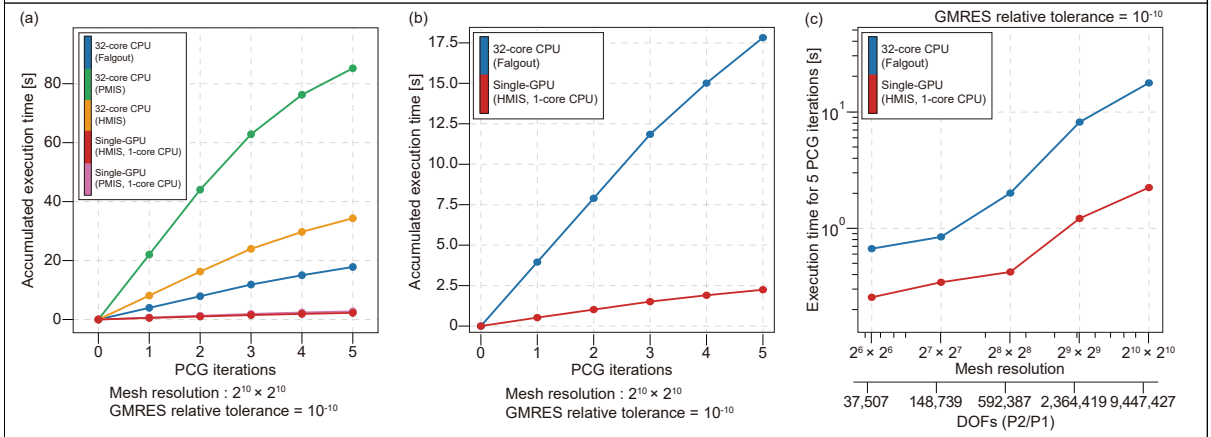
Main point 1-1:
exhaustive comparisons of 1, 2, 4, 8, 16, and 32 CPU cores in various sections doesn't make sense if the focus is GPU-accelerated computations. After the first section, please only show 32 cores.

Authors' reply:

In the original manuscript, comparisons using 1, 2, 4, 8, 16, and 32 CPU cores were repeatedly presented in several result figures. In the revised manuscript, we retain the 1–32 core comparison only in the first performance section to demonstrate CPU scaling behavior. For all subsequent benchmarks, we use only the 32-core CPU as the CPU baseline for comparison with the GPU, as suggested by the reviewer. An example of this revision is shown in the figure below.



Original Figure 2



Revised Figure 2

Main point 1-2 and 1-3:

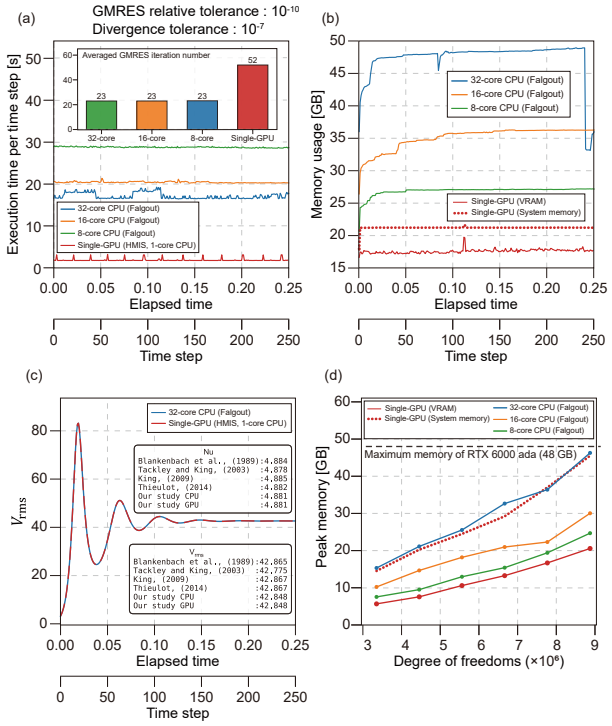
- The authors spend a lot of figures and text on "verification" of CPU vs GPU results even though the same solver, the same matrices, and the same tolerances are used. In fact, it is actually the identical PETSc algorithm that runs (right?). Unless tolerances are not strict enough, the results should not depend on whether they are solved on the GPU (even with different AMG configuration). If they are, it is equally likely to get different results with different solver configurations for two different CPU configurations. Please shorten this!

- My understanding of the focus of this manuscript is CPU vs GPU performance. Maybe a selection of the examples presented is enough to establish the relative performance (especially if the performance behavior is very similar in different examples)

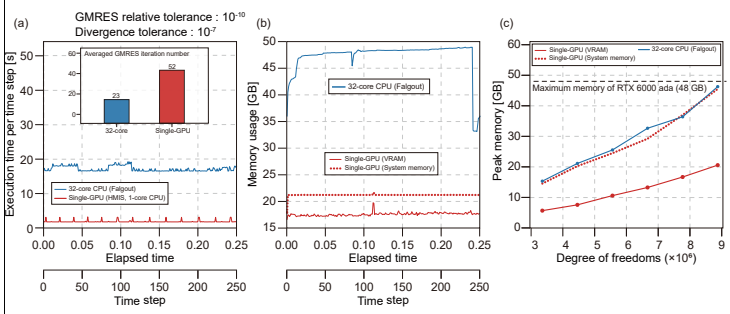
Authors' reply:

We thank the reviewer for this helpful comment and address Main points 1-2 and 1-3 together. The reviewer is correct that the CPU and GPU calculations use the same finite-element discretization and the same Schur-complement PCG algorithm. However, the complete solver configurations are not identical. The GPU implementation of BoomerAMG does not support the Falgout coarsening and SOR smoothing strategies used for the CPU calculations, so different AMG configurations are required for the two backends. Such differences affect the convergence path and iteration count even on the same hardware; for example, the 32-core CPU calculation requires 13 iterations with Falgout coarsening and 25 iterations with HMIS coarsening. With sufficiently strict convergence tolerances, the final CPU and GPU solutions are nevertheless expected to agree. Small differences remain in the reported L_2 errors in Tables 1 and 2, but the differences remain within the discretization-error scale and do not affect the numerical convergence trends.

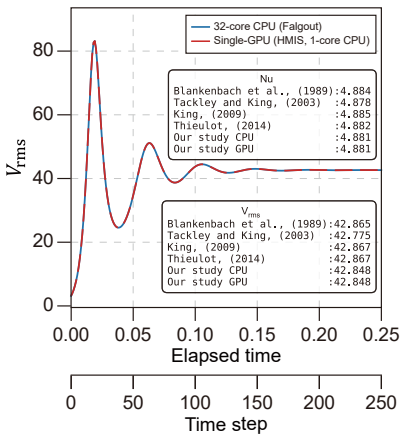
We therefore removed repeated CPU–GPU solution comparisons that only demonstrated agreement between the two backends. The benchmark validation itself was retained. SolCx is compared with its analytical solution, while the sticky-air benchmark remains compared with published reference values. Those comparisons evaluate the numerical implementation rather than CPU–GPU equivalence. For the performance analysis, we retained only cases that provide distinct information on solver behavior or computational workload, such as the comparison of AMG coarsening strategies, the SolCx solver-tolerance analysis, and the 3-D thermo-mechanical convection case.



Original Figure 6



Revised Figure 6

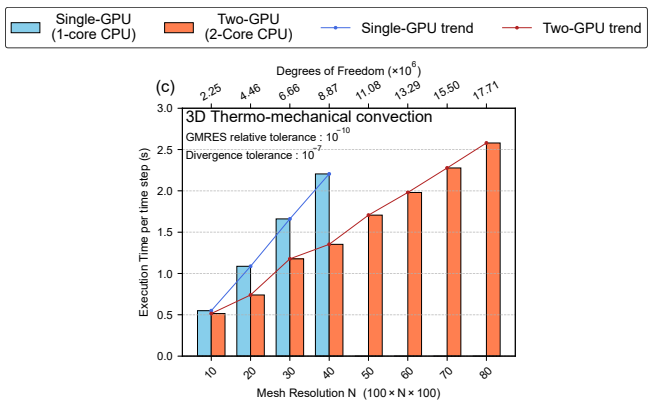


The original Figure 6c has been moved to the Supplementary Material

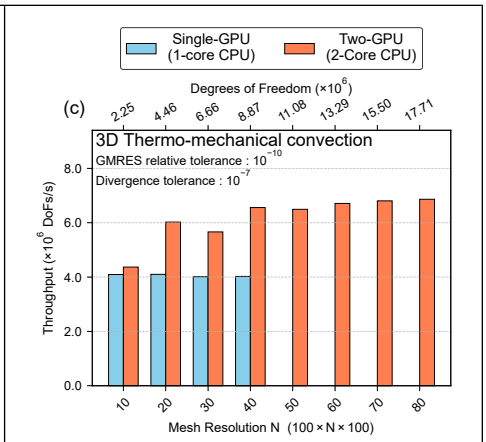
Main point 2:
Solver and performance studies typically include strong scaling and weak scaling plots: while timings for different mesh sizes are reported, comparison with the optimal runtime (time/DoF) based on one data point is missing. GPU publications running on a single GPU typically report throughput like DoFs/s (as you cannot run on different numbers of cores). I think this would be relatively easy to add (most of the data is already collected) and would add a lot to the presentation.

Authors' reply:

We thank the reviewer for this helpful suggestion. We agree that throughput provides a more informative measure of GPU performance than execution time alone for the single- and multi-GPU configurations. In the revised manuscript, we therefore replaced the execution-time results in Figure 14 with throughput (DoFs/s) and added the corresponding time per DoF to the Supplementary Material. The single-GPU throughput remains nearly constant at approximately 4.0–4.1 $\times$ 10^6 DoFs/s over the tested problem sizes, indicating sustained computational efficiency as the problem size increases. The two-GPU throughput is lower for the smallest problems because inter-GPU communication and synchronization overhead constitute a larger fraction of the total cost. Once the problem size exceeds approximately 4.46 $\times$ 10^6 DoFs, the two-GPU throughput increases above 6 $\times$ 10^6 DoFs/s and reaches approximately 1.7 times the single-GPU throughput. These results demonstrate that the single GPU maintains nearly constant throughput, while the two-GPU configuration becomes increasingly advantageous once the computational workload is sufficiently large.



Original Figure 14c and label



Revised Figure 12c and label

Main point 3:
You are implementing a Schur complement solver (PCG on Schur complement with inner GMRES) but I am surprised that you are not clearly establishing the robustness of the outer solver with mesh refinement (number of PCG iterations required until convergence). Can you add a table?

Authors' reply:

We agree with the reviewer's comment. In the original manuscript, we reported the changes in execution time and problem size (i.e., DOFs) with increasing mesh resolution, but we did not directly assess the convergence robustness of the outer PCG solver under different mesh resolutions. In the revised manuscript, we therefore measured the number of outer PCG iterations required to satisfy the same convergence criterion at each mesh resolution. The outer PCG solver converged in two iterations for all tested mesh resolutions from 64^2 to 1024^2, indicating mesh-independent convergence over this range. The results are presented in a new table, including the mesh resolution, the corresponding number of degrees of freedom, and the number of PCG iterations required for convergence. The relatively slow reduction in the pressure L_2 error reflects the pressure discontinuity in the SolCx benchmark and the use of a continuous P_1 pressure approximation. Please see the table below.

Table 1: Mesh-resolution dependence of the PCG solver for SolCx on a 32-core CPU. GMRES relative tolerance is 10^{-10}, and divergence tolerance is 10^{-6}.

Resolution	Stokes DOFs	Execution time (s)	Outer	Velocity L_2 error	Pressure L_2 error
64^2	37,507	0.0679	2	3.2005×10^{-8}	4.3786×10^{-3}
128^2	148,739	0.1506	2	4.0030×10^{-9}	3.0590×10^{-3}
256^2	592,387	0.8070	2	5.0923×10^{-10}	2.1497×10^{-3}
512^2	2,364,419	4.0943	2	7.9689×10^{-11}	1.5152×10^{-3}
1024^2	9,447,427	17.1246	2	2.5796×10^{-11}	1.0696×10^{-3}

Table 2: Mesh-resolution dependence of the PCG solver for SolCx on a single GPU (1-core CPU). GMRES relative tolerance is 10^{-10}, and divergence tolerance is 10^{-6}.

Resolution	Stokes DOFs	Execution time (s)	Outer	Velocity L_2 error	Pressure L_2 error
64^2	37,507	0.0367	2	3.2005×10^{-8}	4.3787×10^{-3}
128^2	148,739	0.0439	2	4.0030×10^{-9}	3.0590×10^{-3}
256^2	592,387	0.1278	2	5.0928×10^{-10}	2.1498×10^{-3}
512^2	2,364,419	0.3383	2	7.9782×10^{-11}	1.5152×10^{-3}
1024^2	9,447,427	1.2556	2	2.5918×10^{-11}	1.0696×10^{-3}

New Tables 1 and 2 in revised manuscript

Main point 4:
You claim a big disadvantage of block solvers is the fact that (F)GMRES does not have a short recurrence and requires restarts and a large number of temporary vectors. I think this is misleading:
- You can use methods with a short recurrence (BiCGStab, IDR(s), ...). In fact, Clevenger, Heister 2021 uses IDR(2) for improved performance for the large simulations.
- You end up using GMRES anyways for the inner solver (and the velocity problem is not much smaller than the whole system, especially for 3d Taylor-Hood). Why not use something else?

Authors' reply:

We appreciate the reviewer's comment, which prompted us to reconsider the scope of our original statement. We agree that the original manuscript was too general in describing restarts and Krylov-vector storage as disadvantages of block-preconditioned solvers. Short-recurrence methods such as BiCGStab and IDR(s) can also be used. For example, Clevenger and Heister (2021) used IDR(2) for large Stokes systems. We have therefore revised the manuscript to clarify that the storage limitation discussed here applies specifically to FGMRES-based block solvers.

We also tested CG as an alternative inner solver because the velocity stiffness matrix is symmetric positive definite. Although CG reduces the Krylov-vector storage relative to GMRES, the reduction in total memory usage was small because the BoomerAMG hierarchy accounts for most of the solver memory. We therefore revised the discussion in Section 5.4 to clarify that the overall memory requirement is governed primarily by the AMG preconditioner rather than by the inner Krylov basis.

Lines 56-59; Section 1,
However, even with restart, FGMRES stores substantially more Krylov basis vectors per cycle than the CG-based approach and restart itself may degrade convergence. The storage limitation applies specifically to FGMRES-based block solvers. In contrast, short-recurrence Krylov methods such as BiCGStab and IDR(s) maintain a fixed storage requirement independent of the iteration count. Clevenger and Heister (2021) used BiCGStab and IDR(s) to solve the full Stokes system in three-dimensional mantle convection simulations.In contrast, CG applied to the pressure Schur complement system requires only a fixed, small number of pressure- and velocity-space vectors~~, offering a clear advantage in GPU environments where VRAM is limited~~in the outer iteration. The inner velocity solve addsseparate Krylov storage.

Lines 627-629; Section 5.4,
In contrast, the PCG solver adopted in this study applies a conjugate-gradient iteration to the pressure Schur complement system. ~~Because it stores~~ The outer iteration stores only a fixed number of vectors, ~~this approach has a smaller memory footprint than block-preconditioned FGMRES,which is advantageous in GPU environments where VRAM capacity is a binding constraint.~~requiring less memory than block-preconditioned FGMRES, which accumulates Krylov vectors as the iteration proceeds. However, the inner velocity subproblem is solved with GMRES and requires additional Krylov-vector storage. As a result, the overall memory saving is smaller than that suggested by the outer iteration alone.

Lines 642-646; Section 5.4,
A block-preconditioned formulation solves the full Stokes system with block preconditioned FGMRES, and a carefully tuned GPU implementation of this strategy may ultimately be faster than the PCG splitting approach. . . . The velocity-only submatrix is symmetric positive definite, allowing CG to replace GMRES for the velocity subproblem. CG stores fewer Krylov vectors than GMRES, whereas the BoomerAMG hierarchy holds most of the solver memory and caps the total saving. Selecting the most effective solver requires a systematic comparison of PCG splitting and block-preconditioned FGMRES in execution time and memory usage.

Execution time and peak VRAM of CG and GMRES across inner solver tolerances for the SolCx benchmark

	CG		GMRES		
Inner rtol	Execution time (s)	Peak VRAM (GiB)	Execution time (s)	Peak VRAM (GiB)	Outer
10^{-4}	2.434	15.24	2.193	15.82	7
10^{-5}	0.829	15.32	0.664	15.99	2
10^{-6}	0.825	15.24	0.874	16.09	2
10^{-7}	0.852	15.35	0.967	16.25	2
10^{-8}	0.925	15.24	1.046	16.39	2
10^{-9}	1.164	15.32	1.190	16.55	2
10^{-10}	1.078	15.32	1.233	16.67	2

Minor comments:

Minor comments 1.
- Assembly on CPU is not strictly required; this is a limitation of the software (see MFEM for example). Please make this clearer.

Authors' reply:

We agree with the reviewer that CPU-side assembly is not inherently required. In our study, CPU-side assembly results from a limitation of the FEniCS implementation rather than a fundamental limitation of GPU architectures. We have revised the manuscript to clarify this point and added a note that other finite-element frameworks, such as MFEM, support GPU-side assembly. Please see the revised text below.

Lines 189-190; Section 3,
With FEniCS GPU acceleration, matrix assembly is performed on CPU cores tied to MPI ranks mapped to GPUs. Although GPU-side matrix assembly has already been developed (Cecka et al., 2011; Markall et al., 2013; Cui et al., 2018), performing CPU assembly remains a limitation of our code implemented in FEniCS 2019.

Lines 539-540; Section 5.1,
Nonlinear Stokes systems with strain-rate-dependent viscosity ~~still~~ require repeated assembly of the stiffness matrix ~~on the CPU, which causes CPU–GPU data-transfer overhead~~.FEniCS 2019 performs each assembly on the CPU, adding a CPU–GPU data transfer at every nonlinear iteration.

Lines 559-561; Section 5.4,
~~Recent work has also explored GPU-accelerated matrix assembly. FEniCSx and Firedrake are adopting CUDA-kernel assembly via projects such as cuda-dolfinx and PyOP2 (Trotter et al., 2023; Rathgeber et al., 2016). Currently, GPU-side assembly remains experimental, with most implementations delivered through extension projects rather than stable support.~~ Recently, finite element codes have incorporated GPU acceleration through global sparse matrix assembly and matrix-free evaluation. GPU execution of assembly kernels is available through cuda-dolfinx in FEniCSx (Trotter et al., 2023) and through PyOP2 in Firedrake (Rathgeber et al., 2016). Matrix-free GPU evaluation is available in deal.II and MFEM (Kronbichler and Ljungkvist, 2019; Anderson et al., 2021). MFEM additionally assembles global sparse matrices on the GPU.

Minor comments 2.
- Abstract: "The GPU is a computing architecture designed for parallelism in vector and matrix operations." - I find this sentence confusing. Are you referring to vectors/matrices used for FEM problems, are you referring to scalar and tensor cores in the GPU, or how parallelism in the GPU works (GPUs are much more generic than

that)? Either way, please clarify and update the abstract.

Authors' reply:

The original sentence in the Abstract was unclear because it could be interpreted as restricting the general GPU architecture to vector and matrix operations. We revised the sentence to clarify that GPUs are general-purpose parallel computing architectures. Their large number of processing units and high memory bandwidth enable efficient handling of data-parallel workloads. Please see the revised text below.

Abstract, first sentence,
~~The GPU is a computing architecture designed for parallelism in vector and matrix operations.~~A GPU couples thousands of lightweight cores with high memory bandwidth. Data-parallel computations with regular memory access patterns can effectively exploit this architecture. To solve the Stokes equations in geodynamics, we apply a preconditioned conjugate-gradient (PCG) method to the pressure Schur complement system.

Minor comments 3.
- L105: You report a Xeon Gold 5320 and a Xeon w5-2465X here. Which one is used for the CPU timings in the paper?

Authors' reply:

We thank the reviewer for pointing this out. Upon rechecking the hardware information, we found that the CPU model number reported in the original manuscript was incorrect, and we apologize for this error. All CPU execution-time evaluations reported in this study were performed using an Intel(R) Xeon(R) Gold 6338 processor (32 cores, 2.00 GHz, 48 MB cache). The Intel Xeon w5-2465X is the host CPU of the GPU workstation and was used only for matrix assembly and other host-side operations; it was not used for the CPU timing measurements. In the revised manuscript, we corrected the CPU model number and specifications. We also clarified the computational configuration used for each processor.

Lines 104-108; Section 2,
All experiments were performed on three machines with different performance characteristics. The first workstation used an Intel Xeon Gold ~~5320~~6338 processor (~~2.20 GHz, 39 MB cache~~2.00 GHz, 48 MB cache, 32 physical cores and 64 threads). The second workstation used an Intel Xeon w5-2465X processor (3.10 GHz, 33.75 MB cache) and a GeForce RTX 6000 Ada GPU with 48 GB of memory. The third workstation used the same processor but was paired with two GeForce RTX 6000 Ada GPUs, each with 48 GB of memory (96 GB in total). All CPU execution times reported in this study were measured on the first workstation, with one MPI rank assigned to each physical core. The second and third workstations were used for the single-GPU and dual-GPU calculations, respectively. The Xeon w5-2465X processor in both systems served only as the host CPU for matrix assembly and other host-side operations.

Lines 675-677; Code and data availability,
The CPU runs were executed on ~~Intel Xeon Gold 5320 and Intel Xeon w5-2465X processors~~an Intel Xeon Gold 6338 processor, while the Intel Xeon w5-2465X served as the host CPU of the GPU workstations; the GPU runs were executed on NVIDIA RTX 6000 Ada (48 GB VRAM) workstations.

Minor comments 4.
- L165: I am not sure about this approach: your GMRES needs to solve very accurately for the outer CG to work reliably, right? A block preconditioner can get by with a spectrally equivalent operator (a single V-cycle, for example). Your GMRES also needs to store a potentially large basis and requires restarts. Why not use some Krylov method that does not require this?

Authors' reply:

We agree with the point raised. In the current implementation, the velocity subproblem is solved to

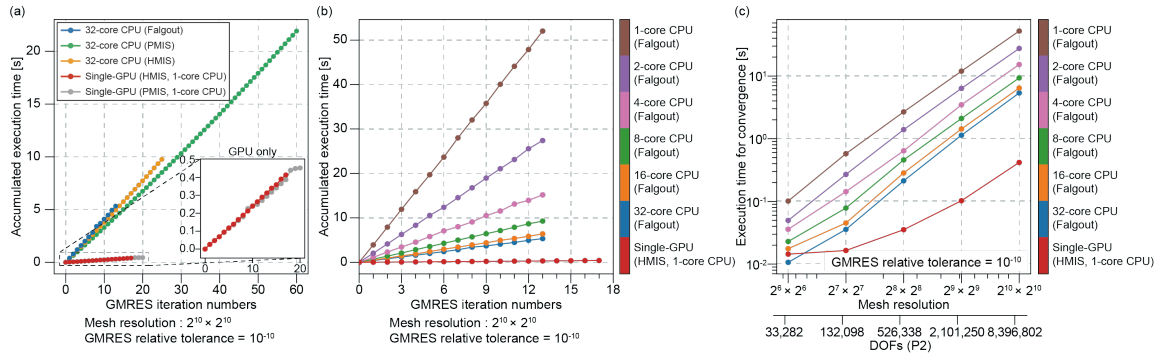
sufficiently tight convergence using GMRES as the inner solver preconditioned with BoomerAMG to ensure reliable convergence of the outer PCG iteration. However, this approach can increase both computational cost and Krylov-vector storage because GMRES requires a growing basis and periodic restarts. Since the velocity stiffness matrix is symmetric positive definite, short-recurrence Krylov methods such as CG can also be considered for the inner solve. We have therefore revised the manuscript to avoid implying that GMRES is a necessary choice. We also clarified the discussion of inner solver (GMRES, CG, or a fixed number of BoomerAMG V-cycles) selection and memory requirements. Please see the revised text below.

Lines 162-163; Section 2,
The algorithm (Equations 7-12) requires only a fixed number of vectors in the outer PCG iteration: approximately four in the pressure space and two in the velocity space. The inner solve of the velocity subproblem executes GMRES with a Krylov basis of its own. Symmetric positive definiteness of the velocity stiffness matrix admits CG in place of GMRES, a short-recurrence method with a basis of only a few vectors. A third treatment applies a fixed number of BoomerAMG V-cycles directly as a spectrally equivalent approximation of K^{-1} within the outer PCG iteration, removing the inner Krylov solver.

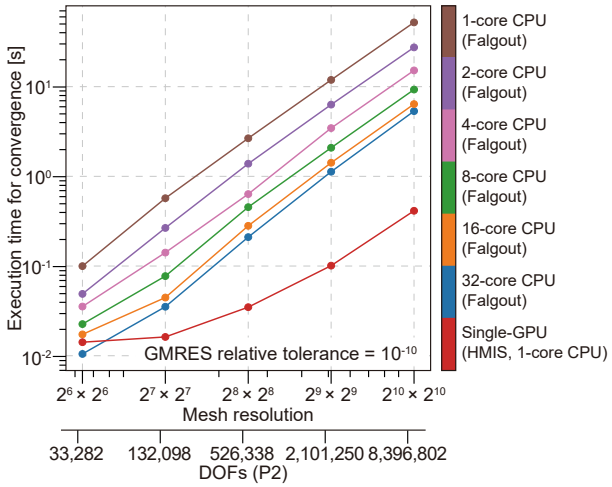
Minor comments 5.
- L264 (Figure 1): The way the data is presented is somewhat unconventional (accumulating time over time). Some of this information might be better represented in a table instead of a figure. I don't see the value of showing CPU time for many different core numbers. The focus of the paper is GPU vs CPU comparison, right? A table with required number of iterations and time to solution would be helpful.

Authors' reply:

Presenting accumulated execution time over iterations in the original Figure 1 made the solver performance unnecessarily difficult to interpret. Therefore, we simplified the figure and reorganized the results into a table. The table lists the number of iterations required for convergence and the total time to solution. Following the reviewer's Main point 1, the comparison across 1, 2, 4, 8, 16, and 32 CPU cores is retained only in this first performance section, where it establishes the CPU scaling behavior and identifies the 32-core configuration as the fastest CPU baseline. Those numbers are now given in the table rather than in the figure, and every subsequent benchmark reports the CPU results for the 32-core configuration alone. Please see the revised figure and the new table.



Original Figure 1



Revised Figure 1

Table 3: Comparison of AMG coarsening strategies on the 32-core CPU and single GPU

Configuration	Coarsening	Iterations	Execution time (s)	GPU speedup
32-core CPU	Falgout	13	5.331	12.9
32-core CPU	HMIS	25	9.756	23.6
32-core CPU	PMIS	60	21.915	53.1
Single GPU (1-core CPU)	HMIS	17	0.413	1.0
Single GPU (1-core CPU)	PMIS	20	0.452	1.1

Table 4: CPU scaling and single-GPU performance

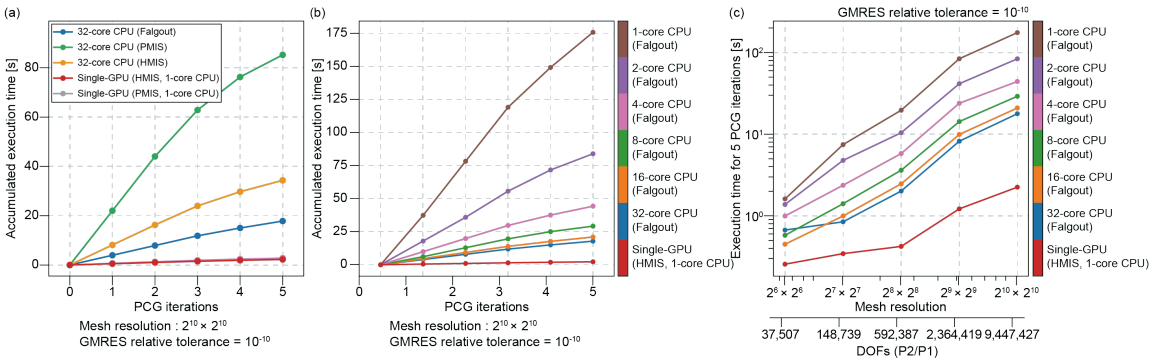
Configuration	Iterations	Execution time (s)	GPU speedup
1-core CPU (Falgout)	13	52.032	126.0
2-core CPU (Falgout)	13	27.400	66.3
4-core CPU (Falgout)	13	15.175	36.7
8-core CPU (Falgout)	13	9.281	22.5
16-core CPU (Falgout)	13	6.391	15.5
32-core CPU (Falgout)	13	5.331	12.9
Single GPU (HMIS, 1-core CPU)	17	0.413	1.0

New Tables 3 and 4 in revised manuscript

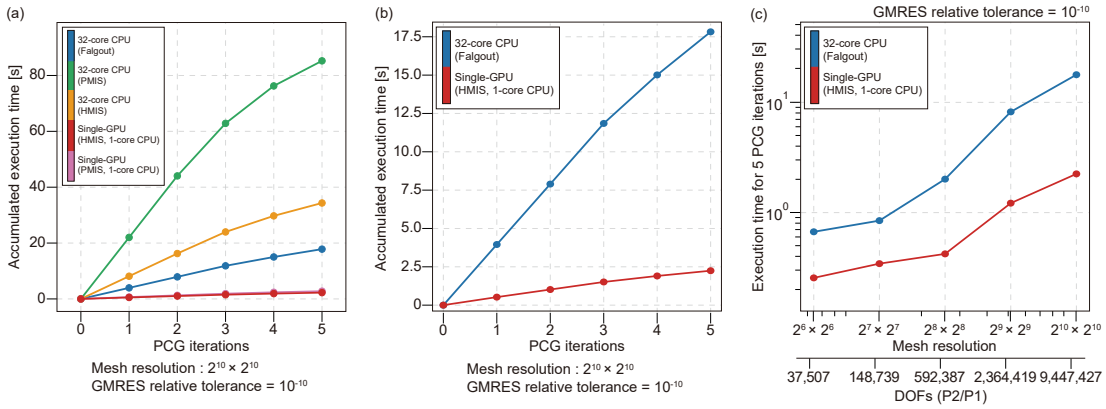
6.- L275 (Figure 2): please remove <32 cores and rescale plots

Authors' reply:

Following the reviewer's suggestion, we removed all CPU results using fewer than 32 cores from Figure 2. The revised figure now uses the 32-core CPU case as the baseline for direct comparison with the single-GPU results. The vertical axis was accordingly re-scaled from 0-180 s to 0-18 s in panel (b), and from 0.26–175 s to 0.26–18 s in panel (c). Please see the figure below.



Original Figure 2



Revised Figure 2

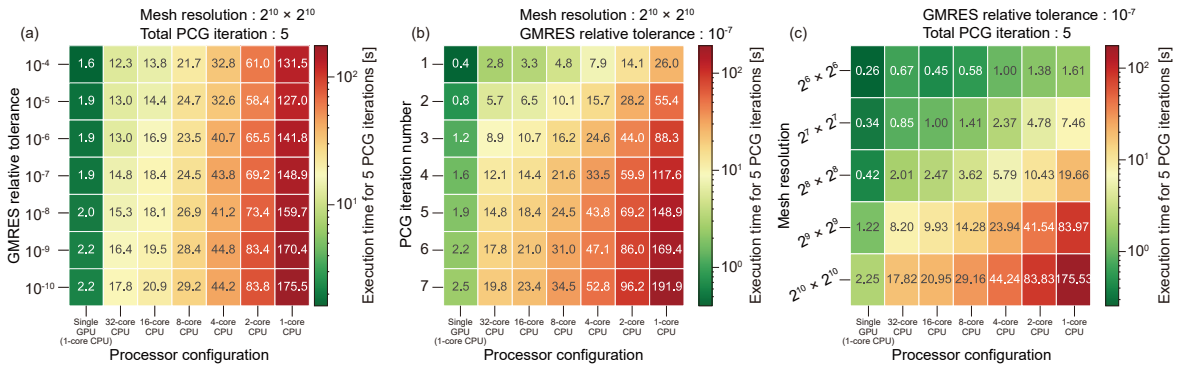
7.- L310 (Figure 4): The plot a) and its conclusion: Of course they become more expensive with tighter inner tolerance but you are ignoring that the required number of PCG iterations likely changes with the inner tolerance (or PCG doesn't converge if inner is too loose?). I don't understand the point of this figure. You should probably pick an outer tolerance for PCG, vary GMRES inner tolerance to find the best time to solution (and report that). All the scaling numbers between CPU (32 cores) and GPU are nearly identical for all data points, which is not surprising. Even if there are trends, your presentation makes it impossible to see. Why do you need to measure the cost for each PCG iteration (b)? Linear increase is expected and probably not worth showing here.
For c): is the number of required PCG iterations resolution independent (do you have an optimal method)? I find the comparison of GPU vs CPU (32 cores) for each problem size the most interesting here (a single or fixed number of PCG iterations is fine for that).

Authors' reply:

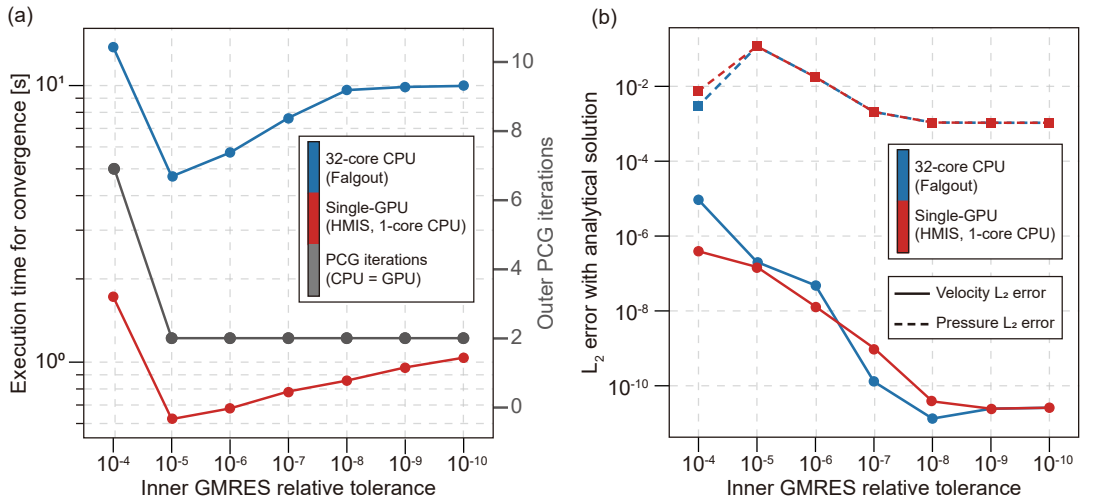
We agree with the reviewer's comment. In the original Figure 4a, we varied only the inner GMRES tolerance while keeping the number of PCG iterations fixed, which did not adequately capture the effect of the inner tolerance on the convergence of the outer PCG solver. In the revised manuscript, we therefore fixed the convergence criterion of the outer PCG solver and varied the inner GMRES tolerance, while evaluating both the number of PCG iterations required for convergence and the total time to solution.

We also agree that the approximately linear increase in execution time with the number of PCG iterations shown in the original Figure 4b is an expected result and provides limited additional insight. We therefore removed original Figure 4b. For the mesh-resolution comparison, we have revised the analysis to focus on the performance difference between the 32-core CPU and GPU.

We also report the number of PCG iterations for each mesh resolution to evaluate the convergence robustness of the outer solver under mesh refinement.



Original Figure 4



Revised Figure 4

8.- Figure 14: What are these lines connecting to one corner of the bars? That looks wrong and confusing. Again, showing throughput might be more helpful here.

Authors' reply:

The lines in the original Figure 14 were added to indicate performance trends. However, the added lines unnecessarily complicated the figure. Thus, we removed the lines in the revised manuscript. As an example, the original and revised versions of Figure 14c are shown below. We also added computational throughput (DoFs/s) as a function of problem size, following the suggestion in Main Point 2, to enable a more direct comparison of single- and multi-GPU performance.

Table S4: Comparison of computational wall times for 3D thermo-mechanical convection simulations across different hardware configurations (GPU, CPU) and ASPECT.

3D thermo-mechanical convection	GPU (two-GPU, 16-core)	GPU (single-GPU, 16-core)	CPU (32-core)	ASPECT (32-core)
Total wall time	169.87 s	239.99 s	2855.62 s	2438.6 s
Assemble Stokes	0.04 s	0.15 s	0.18 s	616 s
Mesh generation	0.85 s	0.74 s	2.73 s	0.15 s
Set up initial conditions	0.00 s	0.00 s	0.04 s	0.12 s
Solve Stokes	139.32 s	200.76 s	2782.39 s	1473 s
Assemble temperature	6.60 s	6.57 s	5.11 s	166 s
Solve temperature	19.35 s	28.18 s	65.56 s	66 s

Original Table S4

Table S4: Comparison of computational wall times for 3D thermo-mechanical convection simulations across different hardware configurations (GPU, CPU) and ASPECT.

3D thermo-mechanical convection	**GPU (two-GPU, 16-core)**	**GPU (single-GPU, 16-core)**	**CPU (32-core)**	**ASPECT (32-core)**
Total wall time	**195.360 s**	**319.069 s**	**1013.739 s**	**1123.000 s**
Initialization / mesh	0.573 s	0.607 s	0.770 s	1.440 s
Setup DoF systems	0.212 s	0.214 s	0.200 s	0.138 s
Setup initial conditions	0.373 s	0.401 s	0.080 s	0.116 s
Setup matrices/KSP	0.019 s	0.013 s	0.041 s	1.060 s
Assemble/update Stokes system	3.356 s	3.401 s	5.083 s	269.000 s
Build Stokes preconditioner	1.439 s	2.618 s	0.082 s	1.850 s
Solve Stokes system	126.620 s	247.352 s	961.005 s	471.000 s
Assemble temperature system	49.941 s	48.552 s	42.726 s	312.000 s
Build temperature preconditioner	0.553 s	0.596 s	0.277 s	6.990 s
Solve temperature system	6.138 s	7.639 s	1.046 s	3.260 s

Revised Table S4

We sincerely thank the reviewer for the careful and constructive evaluation of our manuscript. The comments were helpful in two respects in particular. First, they led us to shorten the Results section and to retain only the comparisons that carry information, which brought the focus back to the CPU–GPU performance itself. Second, they prompted us to report throughput rather than execution time alone, to establish the mesh dependence of the outer PCG iteration count, and to state the limits of our claim on short-recurrence methods more accurately. We are also grateful for the reviewer's attention to detail, which led us to correct the processor model and to make the comparison with ASPECT fairer.

Overall, the revision preserves the central message of the manuscript and presents it more concisely, with clearer evidence for the solver behavior.

With best regards,

Kyeong-Min Lee and Byung-Dal So

References of this response letter

Robert Anderson, Julian Andrej, Andrew Barker, Jamie Bramwell, Jean-Sylvain Camier, Jakub Cerveny, Veselin Dobrev, Yohann Dudouit, Aaron Fisher, Tzanio Kolev, Will Pazner, Mark Stowell, Vladimir Tomov, Ido Akkerman, Johann Dahm, David Medina, and Stefano Zampini. MFEM: A modular finite element methods library. *Computers & Mathematics with Applications*, 81:42–74, 2021. doi: 10.1016/j.camwa.2020.06.009.

Cris Cecka, Adrian J. Lew, and Eric Darve. Assembly of finite element methods on graphics processors. *International Journal for Numerical Methods in Engineering*, 85(5):640–669, 2011. doi: 10.1002/nme.2989.

Thomas C. Clevenger and Timo Heister. Comparison between algebraic and matrix-free geometric multigrid for a Stokes problem on adaptive meshes with variable viscosity. *Numerical Linear Algebra with Applications*, 28:e2375, 2021. doi: 10.1002/nla.2375.

F. Crameri, H. Schmeling, G. J. Golabek, T. Duretz, R. Orendt, S. J. H. Buiter, D. A. May, B. J. P. Kaus, T. V. Gerya, and P. J. Tackley. A comparison of numerical surface topography calculations in geodynamic modelling: an evaluation of the 'sticky air' method. *Geophysical Journal International*, 189(1):38–54, 2012. doi: 10.1111/j.1365-246X.2012.05388.x.

Tao Cui, Xiaohu Guo, and Hui Liu. GPU accelerated finite element assembly with runtime compilation. *arXiv preprint arXiv:1802.03433*, 2018.

B. Hillebrand, C. Thieulot, T. Geenen, A. P. van den Berg, and W. Spakman. Using the level set method in geodynamical modeling of multi-material flows and Earth's free surface. *Solid Earth*, 5: 1087–1098, 2014. doi: 10.5194/se-5-1087-2014.

Martin Kronbichler and Karl Ljungkvist. Multigrid for matrix-free high-order finite element computations on graphics processors. *ACM Transactions on Parallel Computing*, 6(1):2:1–2:32, 2019. doi: 10.1145/3322813.

Graham R. Markall, A. Slemmer, David A. Ham, Paul H. J. Kelly, Chris D. Cantwell, and Spencer J. Sherwin. Finite element assembly strategies on multi-core and many-core architectures. *International Journal for Numerical Methods in Fluids*, 71(1):80–97, 2013. doi: 10.1002/fld.3648.

Florian Rathgeber, David A. Ham, Lawrence Mitchell, Michael Lange, Fabio Luporini, Andrew T. T. McRae, Gheorghe-Teodor Bercea, Graham R. Markall, and Paul H. J. Kelly. Firedrake: Automating the finite element method by composing abstractions. *ACM Transactions on Mathematical Software*, 43(3):24:1–24:27, 2016. doi: 10.1145/2998441.

James D. Trotter, Johannes Langguth, and Xing Cai. Targeting performance and user-friendliness: GPU-accelerated finite element computation with automated code generation in FEniCS. *Parallel Computing*, 118:103051, 2023. doi: 10.1016/j.parco.2023.103051.

Qihang Wú, Shoufa Lin, and André Unger. Modeling multi-material structural patterns in tectonic flow with a discontinuous Galerkin level set method. *Journal of Geophysical Research: Solid Earth*, 128: e2023JB026806, 2023. doi: 10.1029/2023JB026806.