

Response to Reviewer 1

Journal: Geoscientific Model Development

Manuscript Number: EGUSPHERE-2026-2922

Title: DynEarthSol v2.0: An Efficient Explicit Lagrangian Solver for Geodynamics, Surface Processes, and Earthquake-Cycle Dynamics

Authors: Chase J. Shyu, Sungho Lee, Xuesong Ding, Jaeyoon Keum, Eh Tan, Luc L. Lavier, and Eunseo Choi

Reviewer 1

Overall evaluation

Shyu and co-authors present implementations of new algorithms in the DynEarthSol v2.0 software, which are relevant for the modeling of coupled geodynamic and surface processes in crustal-scale models. In particular, they discuss leveraging heterogeneous CPU-GPU architectures, coupling to surface process modeling software, and a new implementation of rate-and-state friction models.

I generally find their manuscript to be well written and the algorithms to be relevant to the field and worth publishing. I have left a number of comments below that I would like the authors to address in order to improve the clarity of the manuscript and the impact of their results. If the authors address these comments I am happy to recommend this manuscript for publication.

Response to the overall evaluation

We thank the reviewer for the positive assessment and the detailed suggestions. We have clarified the finite-element formulation, the CPU-GPU implementation and performance comparisons, and the DES-goSPL coupling procedure. We have also expanded the discussion of the comparison with ASPECT-FastScape and added quantitative error estimates to the RSF constitutive benchmark. The responses below explain the changes made in response to each comment.

R1-Comment 1

Line 16: It would be worth mentioning FD and FV approaches here as well. In general the “memory wall” is more related to the existence of an assembled matrix and sparse-matrix vector multiplications (or Krylov solvers with long recurrence relations), not which particular method is used to discretize the equations.

We appreciate the clarification. We now state that discretization methods that form a large sparse linear system suffer the “memory wall” issue.

Changes in manuscript.

“Traditional implicit Finite Element Methods (FEM) often encounter a “memory wall” as model sizes increase, where the storage requirements for large stiffness matrices exceed the capacity of standard High-Performance Computing nodes.”

→

“Discretization methods forming a large sparse linear system (e.g., implicit finite difference, finite element, or finite volume method) often encounter a “memory wall” as model sizes increase, where the storage requirements can exceed even the capacity of modern High-Performance Computing nodes.”

R1–Comment 2

Line 19: this choice of references is not sufficient to claim that “the field” is shifting to matrix-free, explicit formulations, since they only reference the evolution of FLAC. If you want to insist on the “field” and explicit formulations you should include references to other matrix-free explicit methods, such as Lattice Boltzmann methods (e.g. Mora et al 2020, or other works by the authors or others) and other groups making use of similar pseudo-transient methods as FLAC (de Montserrat et al 2026 or other works). However, more broadly even for implicit methods the push to matrix-free methods has the same justification and momentum (e.g. May et al 2015, Clevenger & Heister, 2021, Ilangovan et al 2026). So generally speaking it is more correct to say the field is pushing towards matrix-free methods, independently of whether they approximate matrix-vector products without assembling and storing the matrix, or whether the method completely avoids the use of matrices.

We agree that the cited works do not support a claim about the field as a whole. However, our intention was narrower: to note that explicit dynamic formulations, which do not assemble a large sparse system, are less exposed to the memory wall than methods that do. We have therefore withdrawn the claim about the field’s direction rather than expanding it, and the revised sentence now simply records the existence of these methods. We have also added pseudo-transient references alongside the dynamic relaxation ones, including de Montserrat et al. (2026). We agree with the reviewer’s broader point about matrix-free implicit solvers, but since the revised text no longer makes a claim about the field, we felt those references would sit unused in a sentence that does not argue for them.

Changes in manuscript.

“To overcome this bottleneck, the field is increasingly shifting toward matrix-free, explicit formulations that offer superior scalability on modern hardware (e.g., Cundall and Board, 1988; Choi et al., 2013).

DynEarthSol (DES) uses an explicit, unstructured finite-element method based on the FLAC algorithm (Cundall and Board, 1988; Poliakov et al., 1993; Lavier et al., 2000).”

→

“However, some methods based on explicit dynamic formulations have a much smaller memory footprint, including dynamic relaxation (e.g., Otter, 1965; Otter et al., 1966; Hodgkins, 1967; Cundall and Board, 1988; Chung et al., 1998; Choi et al., 2013) and pseudo-transient methods (e.g., Kelley and Keys, 1998; Räss et al., 2019, 2022; Alkhimenkov and Podladchikov, 2025; Duretz et al., 2026; De Montserrat et al., 2026).

DynEarthSol (DES) is a dynamic relaxation code in this sense: an explicit, unstructured finite-element implementation of the FLAC algorithm (Cundall and Board, 1988; Poliakov et al., 1993; Lavier et al., 2000). ”

R1–Comment 3

Line 80: Here it would be good to at least mention the stability conditions of the chosen element as well. Linear elements may be a sufficient balance between accuracy and computational efficiency, but depending on their stability they may still not be a good choice for a particular problem. Depending on the exact choice of equations and problem rheology this may or may not be a problem in practice, but it should be discussed in a section about the finite element choice. Also the section could be a bit more clear on which variables are the independent solution variables (I assume force, velocity, and temperature) and which ones are derived (stress, strain, material properties). Also you mention that you use linear basis functions, but from the description I assume they apply to the solution variables, and if stress, strain and material properties are computed at element centroids they are assumed to be cell-wise constant properties, is this correct? If so, please state this in the text. If not, clarify that linear functions are used for all properties.

We agree that this section can be expanded for greater clarity. We have revised the section to address each point: (i) we now state that the linear simplices (P1 elements) used in DES carry element-wise constant volumetric strain and therefore lock as the incompressible limit is approached, and that the nodal mixed discretization of Detournay and Dzik (2006) is adopted as the remedy, which is applied unconditionally in both 2D and 3D; (ii) we now separate the independent solution variables from the derived ones. The reviewer’s assumption was close but not exact: velocity and temperature are the independent unknowns, while nodal force is assembled from the element-wise stress divergence and acceleration follows from the diagonal mass matrix, so both are derived rather than solved for; (iii) we confirm the reviewer’s reading that stress, strain, and material properties are piecewise constant within each element, and now state this explicitly alongside the linear interpolation of the solution variables.

Changes in manuscript.

“DES uses unstructured triangular elements in 2D and tetrahedral meshes in 3D. The basis functions are linear (first-order), providing a balance between accuracy and computational efficiency for large-deformation problems. Force, velocity, and temperature are discretized at nodes, while stress, strain, and material properties are computed at element centroids.”

→

“DES discretizes the domain with an unstructured mesh of P1 elements: linear triangles in 2D and linear tetrahedra in 3D. Velocity and temperature are the independent solution variables and are

interpolated with linear basis functions; nodal forces are assembled from the element-wise stress divergence, and accelerations follow from the nodal assembly of mass. Derived quantities (e.g., stress, strain, strain rate) and material properties (e.g., density, viscosity) are piecewise constant within each element. Constant-strain simplices (i.e., P1 elements) are known to lock volumetrically as the incompressible limit is approached. DES therefore applies the nodal mixed discretization of Detrounay and Dzik (2006) unconditionally in both 2D and 3D, averaging the volumetric strain rate over the elements surrounding each node while the deviatoric part remains element-wise, which relieves the over-constraint from the incompressibility condition. For further details including the explicit time integration scheme, readers are referred to Choi et al. (2013)."

R1–Comment 4

Section 3.2, line 133: This sounds as if the GPU part of the code is Nvidia-hardware specific (or was the Nvidia compiler simply used for the specific systems used for testing?). Please clarify in the text if the code for the accelerator is hardware-agnostic and would run on AMD, Intel or other accelerator hardware as well.

The NVIDIA HPC SDK was the compiler used for the test systems, but the accelerator path is not hardware-agnostic. GPU execution requires an NVIDIA device together with the NVIDIA HPC SDK, because the BVH–KNN remeshing kernel is implemented in CUDA and reached through OpenACC–CUDA interoperability. The dependency is therefore on the CUDA toolchain and not merely on the choice of compiler for the OpenACC directives, and AMD or Intel accelerators are not supported at present.

The host CPU, by contrast, is unconstrained. Both Arm (NVIDIA Grace) and x86-64 (AMD EPYC 7763) hosts are used in the benchmarks of Section 3.4, and the CPU-only build carries no accelerator dependency.

Changes in manuscript.

Section 3.2 now states the scope explicitly: GPU execution requires an NVIDIA device together with the NVIDIA HPC SDK because the BVH–KNN remeshing kernel is written in CUDA, whereas the host CPU is unconstrained, and the CPU-only build has no accelerator dependency. Supporting other accelerators would require a vendor-neutral backend for this remeshing kernel and is named as future work.

R1–Comment 5

Line 189–190: If “profiling confirms” this is the case it would be useful to quantify the amount of wall time spend on CPU vs GPU parts of the algorithm. If this is already computed in the benchmark section later it would be worth referencing the relevant sections here.

The quantities are already computed in Section 3.4, so we now give them at this point in the text and cross-reference the benchmark table. We have also removed the phrase “profiling confirms”, which promised evidence that the manuscript did not present.

For the H100 configuration on regular meshes, the kernel step, which executes entirely on the device, accounts for 94.3% of total wall time at the smallest problem size and 69.5% at the largest, while

the CPU-only output stage accounts for 1.8% and 6.0% respectively. The corresponding figures for the A100 configuration are 93.6% / 64.7% for the kernel step and 2.2% / 5.6% for output.

This is not a clean two-way split, because remeshing, which grows from 2.1% to 21.5% of wall time over the same range, is a hybrid stage whose spatial query and interpolation are GPU-resident while its topological regeneration is CPU-bound. Rather than assert a decomposition that the existing measurements cannot substantiate, we bracket it: at the largest problem size the CPU-side share lies between 6.0% and 27.5%, and the GPU-side share between 69.5% and 91.0%.

On host-device migration we have replaced the measurement claim with the architectural one that supports it, namely that the primary data structures remain resident in Unified Memory for the duration of the time-integration loop, so migration is confined by construction to the CPU-only phases.

Changes in manuscript.

The closing paragraph of Section 3.2.5 now reports the kernel and output shares of wall time for the H100 configuration, brackets the CPU-side share at the largest problem size between 6.0% and 27.5%, cross-references Table B1, and drops the phrase “profiling confirms”.

R1-Comment 6

If I/O becomes a bottleneck is there potential for overlap of CPU and GPU operations by putting I/O into a background thread and let it complete on its own terms, while the main computation loop continues?

We agree that the potential exists, but we did not adopt it, for reasons of memory safety rather than of feasibility. Delegating output to a background thread requires one of two things. Either the CPU-side buffers must remain untouched until the write completes, which makes memory management considerably more fragile and makes it easy to introduce silently incorrect output, particularly in a collaboratively developed code where a contributor need not know that a buffer is still being written. Or the buffers must be duplicated for the writer, which doubles the memory footprint of the fields being output. Since the accessible problem size in DES is already bounded by single-node memory, that is a costly trade.

The attainable gain is also bounded. Across every configuration benchmarked here, output accounts for at most 6.9% of total wall time, against the 21.5% consumed by remeshing in the largest H100 case. Remeshing is therefore the more valuable target, and it is the one named in the conclusions. We intend to revisit asynchronous output once the feature set of DES has stabilized.

Changes in manuscript.

A paragraph was added to the end of Section 3.3 recording that asynchronous output was considered but not adopted, giving the memory-safety and memory-footprint reasons, noting that output is at most 6.9% of wall time so the attainable gain is bounded, and stating that the option will be revisited once the feature set has stabilized.

R1–Comment 7

Line 203: For the benchmarked GPU systems, please also specify in the manuscript which CPUs were used on those systems, as it affects the hybrid CPU/GPU parts of your algorithm. From descriptions in the text below it seems like you used nodes that combined Grace CPU with H100 GPUs, and Epyc 7763 CPUs with A100 GPUs, is this correct? This may affect some of the A100 vs H100 performance comparison (the ones with hybrid algorithms), as the Epyc 7763 is slower than the Grace CPU.

The reviewer’s inference is correct. The four configurations are the NVIDIA Grace CPU with 72 cores, an H100 hosted by an NVIDIA Grace CPU with 72 cores, the AMD EPYC 7763 with 64 cores, and an A100 hosted by an AMD EPYC 7763 with 32 cores. The Grace and H100 configurations were run on the Vista cluster and the EPYC 7763 and A100 configurations on the Lonestar6 cluster, both at the Texas Advanced Computing Center, so the pairings follow from the node designs rather than from a choice on our part.

We also note an asymmetry that the original text did not make visible: the A100 host was run with 32 cores whereas the CPU-only EPYC baseline used 64. Both effects push in the same direction, and we agree they qualify the A100-versus-H100 comparison for those stages that are hybrid rather than purely device-side. This matters most for remeshing, whose topological step is CPU-bound, so the reported factor-of-three gap between A100 and H100 remesh time is a machine-level rather than a GPU-level result. The purely CPU-resident kernel step is far less affected.

Changes in manuscript.

Section 3.4 now names the four configurations with their host CPUs and core counts, identifies the two clusters they were run on, and states that comparisons of the hybrid stages between the H100 and A100 configurations reflect node-level rather than GPU-level differences. A corresponding caveat was added where the factor-of-three remesh gap is reported.

R1–Comment 8

Line 205: “identical configuration, except for mesh and domain sizes” is an unfortunate formulation in this context, because of course if the mesh and domain sizes change, the comparison is no longer fair. I think what you mean is that you performed the identical scaling tests (with variable mesh and domain sizes) on different hardware. Please clarify the text.

Agreed, the original phrasing was ambiguous. We meant that the same suite of scaling tests, spanning the full range of mesh and domain sizes, was executed on every hardware platform, and that comparisons across hardware are always made at matched problem size.

Changes in manuscript.

The opening of Section 3.4.1 now reads that an identical suite of scaling tests was executed on each hardware platform, and that all cross-hardware comparisons are made at matched element count.

R1–Comment 9

Line 217: Grace (Grace) probably should be Grace (CPU)? Also the relevant information for the power law is the exponent of the power law (is it linear, less than linear, something else?), please add it to the text.

“Grace (Grace)” was a typographical error and is corrected to “Grace CPU”. On the exponent, we fitted $T \propto N^\alpha$ by least squares in $\log_{10}$ – $\log_{10}$ space to the eight regular-mesh points of Table B1. For total execution time on the Grace CPU, $\alpha = 1.01$ with $R^2 = 0.994$, that is, essentially linear in element count across the full range from 3.9×10^4 to 5.0×10^6 elements. This is the expected result for an explicit solver whose per-step cost is proportional to the number of elements and whose step count is fixed.

Changes in manuscript.

The sentence now gives the fitted exponent, $\alpha = 1.01$ with $R^2 = 0.994$, and states that the scaling is essentially linear across the full range. The fitting procedure is stated once in Section 3.4.1.

R1–Comment 10

Line 218: outperformed the Grace $\rightarrow$ outperformed the Grace CPU.

Corrected as suggested. We also checked the rest of Section 3.4 and fixed the same bare “the Grace” and “the EPYC” usage wherever it referred to the hardware rather than to the vendor.

Changes in manuscript.

The sentence now reads “outperformed the Grace CPU” and “surpassed the EPYC 7763”. The same correction was applied throughout Sections 3.4.2 and 3.4.3.

R1–Comment 11

Line 229: Again the important information is the (approximate) exponent of the power law and to which model sizes the performance scales, please state it in the text.

Added, using the same log–log least-squares fit. For the per-step kernel duration on regular meshes the Grace CPU gives $\alpha = 1.00$ with $R^2 = 0.993$ across the full range, which is linear, as expected when the kernel cost is proportional to element count. The EPYC 7763 gives $\alpha = 0.80$ over the full range but with a visibly poorer fit, $R^2 = 0.969$, because the four smallest cases are not yet throughput-limited. Restricted to the largest four sizes, from 6.3×10^5 to 5.0×10^6 elements, it gives $\alpha = 1.11$ with $R^2 = 0.998$. We therefore quote the asymptotic exponent and state the range over which it holds, rather than a whole-range number that misrepresents the small-size behaviour. For completeness, the GPU kernel exponents are sublinear at small sizes and approach linearity asymptotically, 0.68 over the full range and 0.96 over the largest four for the H100, and 0.67 and 0.92 respectively for the A100, which is the same launch-overhead effect that produces the speedup crossover.

Changes in manuscript.

Section 3.4.2 now gives the fitted kernel-step exponents, $\alpha = 1.00$ for the Grace CPU over the full range and $\alpha = 1.11$ for the EPYC 7763 over the largest four sizes, and notes that below that range the EPYC 7763 is not yet throughput-limited.

R1–Comment 12

Line 227: That the EPYC 7763 underperformed the Grace CPU is not surprising given their theoretical performance comparison, memory bandwidth and their relative age (2021 vs 2023). Maybe that is worth mentioning to avoid the impression that AMD Epycs generally underperform the Grace CPU (e.g. the Epyc 9965 from 2024/2025 is reported to have several times the performance of the 7763, and significantly more than the Grace CPU, so the comparison would likely look different). At least specify the CPU model here again.

We agree, and we did not intend the comparison to read as a general statement about AMD server CPUs. The part named in the benchmark is the EPYC 7763, a 2021 Zen 3 part, and the Grace CPU it is compared against is a 2023 part with substantially higher memory bandwidth, which is the relevant figure of merit here since the DES kernels are bandwidth-bound rather than FLOP-bound. The comparison should be read as one between two specific parts of different generations, not between two vendors.

Changes in manuscript.

The sentence now names the part, “the EPYC 7763 consistently underperformed the Grace CPU”. A sentence was added noting the 2021 versus 2023 generational difference and the memory-bandwidth gap, and stating that a current-generation server CPU would be expected to change the comparison substantially. The part number was also added to the Table 2 header and the Figure 3 caption.

R1–Comment 13

Fig. 3: Please double-check that panels b and e are not just accidental copies from panels a and d. They look identical to me, but should not be identical, as a and d should be a sum of all computational steps.

We have checked this and the panels are genuinely distinct, with no panel duplicated. The resemblance is real but it is a physical result rather than a plotting error, and the original figure did nothing to signal that.

Panels (a) and (b) plot different quantities on different axes. Panel (a) is cumulative wall-clock time in minutes over the whole run, roughly 1 to 500 min, whereas panel (b) is the per-step kernel duration in seconds, roughly 10^{-3} to 4×10^{-1} s. Their shapes are nearly proportional because the kernel step accounts for 92 to 98% of total wall time in every CPU-only configuration.

Panels (d) and (e) are closer still, for the same reason. At maximum core count the kernel speedup exceeds the total speedup by 2.4% at 39,375 elements rising to 5.5% at 315,000 elements on the Grace CPU, and by up to 3.1% on the EPYC 7763. The one case where the two coincide is the smallest EPYC case, where the kernel fraction reaches 98% and the difference falls to 0.1%, which

is presumably where the panels look most alike. The gap widens with problem size because the remesh fraction grows, which is the Amdahl’s-Law argument of Section 3.4.2, so the fact that panels (d) and (e) differ is the evidence for it.

Changes in manuscript.

The Figure 3 caption now notes that panels (a) and (b), and likewise (d) and (e), are similar in shape but not identical, gives the 92 to 98% kernel fraction as the reason, and states that the kernel speedup in (e) exceeds the total speedup in (d) by 2 to 6%. The corresponding figures were added to Section 3.4.4.

R1–Comment 14

Section 3.4 in general: One of the main challenges of GPU acceleration for HPC modeling software is the parallelization across distributed memory parallel systems. This is not mentioned here and all tests are performed on one server node, so I assume that DES is purely parallelized for shared memory parallelization? That would be worth pointing out, as it is an important consideration for potential users.

Correct, and worth stating plainly. DES v2.0 is parallelized for shared memory only. Multi-core CPU execution uses OpenMP and accelerator execution uses OpenACC, both within a single node and a single address space, and there is no MPI layer and no domain decomposition across distributed memory. Accelerated runs moreover use a single GPU device, and multi-GPU execution within a compute node is not supported either. The largest tractable problem is therefore bounded by the memory of one compute node, or of one GPU device plus Unified Memory oversubscription, and all benchmarks reported here are single-node by necessity rather than by choice. We agree this is important information for prospective users and that its absence was an omission.

Changes in manuscript.

A paragraph was added at the start of Section 3.4 stating that all benchmarks were run on a single compute node, that DES v2.0 is parallelized for shared memory only with no distributed-memory domain decomposition, that accelerated execution uses a single GPU device, and that the results should be read as intra-node scaling. The conclusions now name distributed-memory parallelism as the principal remaining constraint on scalability alongside the CPU-bound remeshing stage.

R1–Comment 15

Line 344: This is interesting, it sounds like DES time steps are generally equal or smaller than GoSPL time steps. In other cases of coupling (e.g. ASPECT with FastScape or Badlands with Underworld) generally the tectonic time step is (often significantly) larger than the surface evolution time step. Is this due to the explicit DES time step? Would the code work correctly if the DES time step is ever significantly larger than the GoSPL time step or is the DES time step to be limited to at most one GoSPL time step?

The DES time step is generally smaller than a typical goSPL step because of the explicit dynamic formulation of DES. However, the coupling does not depend on this ordering. A coupling event is triggered when the accumulated time reaches the specified coupling interval, and goSPL is then

advanced by that accumulated interval; nothing in the coupling procedure assumes that the interval spans multiple DES steps. The DES step is therefore not limited to at most one goSPL step. The case of a single DES step reaching the coupling interval ($N=1$) was accounted for in the design, although we have not exercised it in a test. Since goSPL uses an implicit solver, a long coupling interval poses no stability issue in itself; the choice of interval should be guided by the time scales of the processes being modeled.

Changes in manuscript.

In Sec. 4.2.2 Two-way coupling algorithm

“The coupling imposes no ordering between the DES time step and goSPL’s internal time step. A coupling event is triggered when the accumulated time reaches the specified interval, and goSPL is advanced by that accumulated interval regardless of how many DES steps it comprises. The case of a single DES step spanning the full interval is allowed by the design.”

R1–Comment 16

Line 361: How is the time-averaged $\Delta \mathbf{x}$ over the coupling period computed? Is there a back up of the node position at the beginning of every coupling period stored? If so, who owns this backup (DES or goSPL)? And how does this work if the DES grid is remeshed during the coupling period? This might be worth adding to the description of this step.

The time-averaging over a coupling period occurs solely within DES. DES assumes the changes over the period is sufficiently slow approximating them as linear in time. Thus, it stores surface node coordinates only at the beginning of the period and use it at the end of it.

When remeshing changes the number of top-surface nodes, the averaging is skipped and the instantaneous DES velocity is used instead. Because the fallback is triggered by the node count alone, a remeshing operation that reorganized the surface nodes without changing their number would not trigger it, and the averaged velocity would be computed across inconsistent node sets. Such a case is extremely rare in practice, since remeshing almost always alters the surface node count in a tectonic modeling. We state this limitation in the revised manuscript and plan to replace the node-count test with a direct check of surface-node correspondence.

Changes in manuscript.

“Velocities are computed as time-averaged displacements $\Delta \mathbf{x} / \Delta t$ over the coupling interval, falling back to instantaneous velocity on the first coupling event.”

→

“Velocities are computed as time-averaged displacements $\Delta \mathbf{x} / \Delta t$ over the coupling interval, falling back to the instantaneous velocity on the first coupling event *and whenever remeshing changes the number of top-surface nodes since the displacement record is then no longer defined on a fixed node set. Because this fallback is triggered by the node count alone, a remeshing operation that reorganized the surface nodes without changing their number would not trigger it, and the time-averaged velocity would be computed across inconsistent node sets. Such a case is extremely rare in practice, as remeshing almost always alters the surface node count. We plan to replace the node-count test with*

a direct check of surface-node correspondence in a future release.”

R1–Comment 17

Line 380: You write a single IDW pass is necessary (from goSPL to DES), but step 1 describes that velocity is (optionally) transferred from DES to goSPL via IDW, which would make that 2 IDW passes. This seems inconsistent. Is velocity transfer usually not required and therefore in a typical model only one IDW pass is necessary?

The reviewer is correct that the statement was imprecise. The advantage we intended to describe concerns the interpolation of elevation: because Δh is computed entirely within goSPL’s native mesh, absolute elevations are never interpolated between the two meshes, and only one IDW pass of elevation data (goSPL to DES) is needed per coupling event. When velocity coupling is enabled, which is the case by default, a second IDW pass transfers DES surface velocities to the goSPL mesh. That pass carries a different field and does not affect the cancellation of interpolation-smoothing errors in the elevation transfer. We have revised the sentence accordingly.

Changes in manuscript.

“...only a single IDW pass (from goSPL to DES) is required per coupling event.”

→

“...only a single IDW pass of elevation data, from goSPL to DES, is required per coupling event. During the velocity coupling enabled by default, DES surface velocities are transferred to the goSPL mesh in a separate IDW pass (step 1 above), which involves a different field and does not reintroduce elevation-smoothing error.”

R1–Comment 18

Line 382: It would be good to mention if the size of the padded area sets an upper limit on the maximal extensional strain a DES model can accumulate during the model runtime. Or can the goSPL domain be reset after a certain amount of extension has accumulated?

The goSPL mesh is generated once at initialization, covering the initial DES surface extent plus a padding margin (10% by default), and is neither re-centered nor regenerated during the run. The padding therefore sets an upper bound on the extension a coupled model can accumulate. There is currently no check for surface nodes leaving the padded domain: if they do, Δh is interpolated from a one-sided or distant neighborhood and the resulting elevation change is spurious, without any warning being issued. Currently, users should exercise caution to decide the padding zone size for the total extension expected over the intended runtime.

Resetting the goSPL mesh mid-run is not currently supported. It is a design decision to let goSPL accumulate its drainage-network state continuously: a reset would require transferring that state to the new mesh.

Changes in manuscript.

“The goSPL mesh extends a user-specified fraction beyond the DES domain boundary on all four sides (default: 10%). This buffer accommodates the lateral displacement of DES surface nodes that occurs during extension: without padding, nodes that migrate outside the goSPL mesh boundary would produce extrapolation errors. The IDW interpolation of Δh naturally handles nodes anywhere within the padded domain.”

→

“The goSPL mesh extends a user-specified fraction beyond the DES domain boundary on all four sides (default: 10%). This buffer accommodates the lateral displacement of DES surface nodes that occurs during extension. *The IDW interpolation of Δh handles nodes anywhere within the padded domain, but the padding also sets an upper bound on the extension a coupled model can accumulate. The goSPL mesh is generated once at initialization and is neither re-centered nor regenerated during the run. As a result, surface nodes that migrate beyond the padded domain are not detected, and Δh is then interpolated from a one-sided or distant neighborhood, giving a spurious elevation change without a warning. The padding fraction should therefore be chosen for the total extension expected over the intended runtime.*”

“A boundary buffer of 5% on each edge of the goSPL mesh domain prevents extrapolation errors when nodes move during simulation.”

→

“A boundary buffer of 10% on each edge of the goSPL mesh domain *keeps DES surface nodes surrounded by goSPL nodes as they displace during the simulation, so that Δh is interpolated from a neighborhood enclosing each node.*”

R1–Comment 19

Line 418: “between” → “compared to” the DES-only case?

We have removed this sentence. The mesh resolution is now stated explicitly in Sec. 4.3 Test case: Gaussian fault evolution that $h_{\min} \approx 0.5$ km under adaptive refinement.

R1–Comment 20

Line 455: This section needs a sentence summarizing the result of the comparison. Is the comparison between DES-goSPL and ASPECT-FASTSCAPE satisfactory? Are the remaining differences likely caused by small differences in model setup (e.g. different bottom boundary condition and/or minor implementation differences)? Or may they be caused by some fundamental disagreement? I would be particularly curious about the authors opinion on the different roughness of the topography between the two models. Are these models run on similar resolutions? Is the higher roughness of the DES-goSPL model caused by the less structured mesh of DES, or does ASPECT-Fastscape seem to be more diffusive? I understand that a conclusive answer to these questions may be beyond the scope of the paper.

Is the comparison between DES-goSPL and ASPECT-FASTSCAPE satisfactory?

We consider the comparison satisfactory, as both coupled pairs produced consistent results. First, for identical brittle rheology parameters (strain-softening plasticity), the two models developed the same faulting mode, with only minor differences (discussed below). In both, a primary normal fault formed and evolved into a large-offset, low-angle fault that exerted first-order control on the surface topography. Second, the two surface topographies agree in showing a clear imprint of stream incision. In both models, however, surface processes remained secondary to tectonic faulting in controlling the overall topographic evolution.

Are the remaining differences likely caused by small differences in model setup (e.g. different bottom boundary condition and/or minor implementation differences)? Or may they be caused by some fundamental disagreement?

The remaining differences in the surface topography appear to stem from a major implementation difference: element and mesh type. We detail our reasoning below in the response regarding the greater roughness of the DES-goSPL model.

Here, we clarify that the different basal treatments do not represent a substantive difference between the two models. The Winkler foundation used in DES and the low-viscosity layer used in ASPECT play a mechanically identical role. A Winkler foundation is a local, instantaneous restoring pressure proportional to deflection, which is precisely the response of a substratum that flows freely enough to offer no resistance of its own. In the ASPECT model the 10 km-thick 10^{18} Pa s layer beneath the brittle layer is approximately inviscid, and the shear stress it transmits is a negligible fraction of the yield stress of the overlying brittle layer. The lithostatic traction prescribed at the base of the ASPECT domain fixes the reference pressure while leaving the boundary open, so that material may enter and leave the low-viscosity layer. The two models therefore impose the same basal support.

I would be particularly curious about the authors opinion on the different roughness of the topography between the two models.

Are these models run on similar resolutions?

Yes. The tectonic meshes are adaptive, so element size varies in space and time: at 0.48 Myr, elements ranged from about 500 m within shear bands and 1200 m to 2500 m outside in DES; and from 1250 m to 2500 m in ASPECT. The coupled surface process codes, by contrast, use fixed grids throughout, at resolutions of 500 m (goSPL) and 625 m (FASTSCAPE). Since the two surface process grids are fixed and of comparable resolution, the difference in topographic roughness is unlikely to originate there; we instead attribute it to the tectonic meshes, as discussed below.

Is the higher roughness of the DES-goSPL model caused by the less structured mesh of DES, or does ASPECT-Fastscape seem to be more diffusive?

We suspect the difference originates in the tectonic meshes. The two surface process models used identical hillslope diffusivity and fluvial incision parameters, so the smoother topography in ASPECT-FASTSCAPE cannot be attributed to greater diffusivity in FASTSCAPE itself. Instead, the DES mesh consists of irregular linear tetrahedra, whose facets have less coherent orientations on the surface-exposed fault scarp. In contrast, the ASPECT mesh remains locally structured under refinement and its Q2-Q1 elements represent localized shear deformation as a smoother band. These differences appear to explain the rougher scarp in DES-goSPL. A direct cross-check would settle the question, but it is not feasible because the element types are not interchangeable between

the two codes.

We also note a long-lasting effect in the coupled DES simulation: early roughness of the main fault scarp promoted incision, which in turn maintained that roughness. In the standalone DES simulation, by contrast, only hillslope smoothing acted on a similarly rough initial scarp, which therefore became smoother over time.

Changes in manuscript.

Based on these results, we have added the following summarizing sentence at the end of the section:

“Overall, the two coupled models agree at first order: both develop a large-offset, low-angle normal fault that dominates the topographic evolution, and in both the stream-incision signature is clearly present but subordinate to tectonic faulting, with the differences confined to secondary faulting and the resulting topographic roughness.”

Also our reasoning on the origin of the difference in secondary faulting and the resultant difference in roughness has been added to Sec. 4.3 Test case:

“Neither model carries a regularization that sets an internal length scale: the plastic damper available in ASPECT (Bangerth et al., 2024) was not enabled in the calculation shown here, and DES has no equivalent. The difference in secondary faulting is therefore unlikely to reflect regularization in one code and not the other. We speculate instead that it is attributable to the element type and mesh structure: i.e., irregular linear tetrahedra in DES against the locally structured, higher-order (Q2-Q1) elements in ASPECT, which represent localized shear deformation as a smoother band.”

R1–Comment 21

Line 592: compared with analytical solution → compared with an analytical solution.

The sentence has been revised.

Changes in manuscript.

Section 5.3.1: the statement now reads “This agreement verifies the elastoplastic return mapping and the RSF stress and state updates under simple shear.”

R1–Comment 22

Line 622: I appreciate this benchmark case with analytical solution, which forms a great basis for intercode comparisons! That being said, “are visually indistinguishable” can only be the very first step in a comparison between numerical and analytical benchmark solution. Since you have both solutions available, could you (a) at a minimum list the typical (or final) quantitative relative error between numerical and analytical solution, and (b) consider if it is worth plotting the relative error for the different cases in Fig. 10 in addition to the absolute stress values? That would make it easier to use this benchmark in the future to compare accuracy across different software implementations.

We added pointwise relative-error panels beneath the stress panels in Fig. 10b–d. The table below reports the time-mean and final errors for all nine cases.

We calculate $e(t) = |\bar{\sigma}_{xy}^{\text{DES}} - \bar{\sigma}_{xy}^{\text{ref}}|/\bar{\sigma}_{xy}^{\text{ref}}$, where $\bar{\sigma}_{xy}$ is the mean absolute shear stress of the two elements. The stress-free initial state is excluded because the relative error is undefined there.

Case	Mean error (%)	Final error (%)
Elastoplastic, $\phi = 30^\circ$	5.94×10^{-4}	9.13×10^{-4}
Elastoplastic, $\phi = 20^\circ$	6.67×10^{-4}	1.06×10^{-3}
Elastoplastic, $\phi = 10^\circ$	7.12×10^{-4}	1.16×10^{-3}
Steady RSF, $V_0 = 10^{-6} \text{ m s}^{-1}$	6.62×10^{-4}	1.05×10^{-3}
Steady RSF, $V_0 = 10^{-5} \text{ m s}^{-1}$	5.81×10^{-4}	8.88×10^{-4}
Steady RSF, $V_0 = 10^{-4} \text{ m s}^{-1}$	4.94×10^{-4}	7.21×10^{-4}
Aging RSF, $D_c = 10^{-3} \text{ m}$	5.83×10^{-4}	8.87×10^{-4}
Aging RSF, $D_c = 3 \times 10^{-3} \text{ m}$	5.91×10^{-4}	8.91×10^{-4}
Aging RSF, $D_c = 10^{-2} \text{ m}$	6.00×10^{-4}	9.13×10^{-4}

The largest pointwise relative error is $1.16 \times 10^{-3}\%$. The residual difference arises from the small-strain approximation in the reference solution, as explained in Section 5.3.1 and Appendix E.

Changes in manuscript.

Section 5.3.1 reports the error range, Figure 10 includes the relative-error panels, and Appendix E gives the reference solutions and error definition.

R1–Comment 23

Line 674: “can handle the earthquake with ...” either “an earthquake ...” or “can handle earthquakes ...”.

The sentence was removed when the earthquake-cycle discussion was revised.

Changes in manuscript.

Section 5.4: the passage containing the sentence was removed.

R1–Comment 24

Line 675: “elastic wave” $\rightarrow$ “elastic waves”.

The sentence was removed when the earthquake-cycle discussion was revised.

Changes in manuscript.

Section 5.4: the passage containing the sentence was removed.

References supplied by the reviewer

- Peter Mora, Gabriele Morra, David A Yuen, A concise python implementation of the lattice Boltzmann method on HPC for geo-fluid flow, *Geophysical Journal International*, Volume 220, Issue 1, January 2020, Pages 682–702, <https://doi.org/10.1093/gji/ggz423>

- de Montserrat, A., Aellig, P. S., Schuler, C., Navarrete, I., Räss, L., Fuchs, L., ... & Dominguez, H. (2026). JustRelax.jl: A Julia package for geodynamic modeling with matrix-free solvers. *Journal of Open Source Software*, 11(118), 9365.
- May, D. A., Brown, J., & Le Pourhiet, L. (2015). A scalable, matrix-free multigrid preconditioner for finite element discretizations of heterogeneous Stokes flow. *Computer Methods in Applied Mechanics and Engineering*, 290, 496–523.
- Ilangoan, P., Kohl, N., & Mohr, M. (2026). Highly scalable geodynamic simulations with HyTeG. *Geoscientific Model Development*, 19(4), 1455–1472.
- Clevenger, T. C., & Heister, T. (2021). Comparison between algebraic and matrix-free geometric multigrid for a Stokes problem on adaptive meshes with variable viscosity. *Numerical Linear Algebra with Applications*, 28(5), e2375.

References

- Alkhimenkov, Y. and Podladchikov, Y. Y.: Accelerated Pseudo-Transient Method for Elastic, Viscoelastic, and Coupled Hydromechanical Problems with Applications, *Geoscientific Model Development*, 18, 563–583, <https://doi.org/10.5194/gmd-18-563-2025>, 2025.
- Bangerth, W., Dannberg, J., Fraters, M., Gassmoeller, R., Glerum, A., Heister, T., Myhill, R., and Naliboff, J.: ASPECT: Advanced Solver for Planetary Evolution, Convection, and Tectonics, User Manual, <https://doi.org/10.6084/m9.figshare.4865333>, 2024.
- Choi, E., Tan, E., Lavier, L. L., and Calo, V. M.: DynEarthSol2D: An efficient unstructured finite element method to study long-term tectonic deformation, *Journal of Geophysical Research: Solid Earth*, 118, 2429–2444, <https://doi.org/10.1002/jgrb.50148>, 2013.
- Chung, W., Cho, J., and Belytschko, T.: On the Dynamic Effects of Explicit FEM in Sheet Metal Forming Analysis, *Engineering Computations*, 15, 750–776, <https://doi.org/10.1108/02644409810231880>, 1998.
- Cundall, P. A. and Board, M.: A microcomputer program for modeling large strain plasticity problems, in: *Numerical Methods in Geomechanics*, edited by Swoboda, G., pp. 2101–2108, Balkema, Rotterdam, 1988.
- De Montserrat, A., Aellig, P. S., Schuler, C., Navarrete, I., Räss, L., Fuchs, L., Kaus, B. J. P., and Dominguez, H.: JustRelax.Jl: A Julia Package for Geodynamic Modeling with Matrix-Free Solvers, *Journal of Open Source Software*, 11, 9365, <https://doi.org/10.21105/joss.09365>, 2026.
- Detournay, C. and Dzik, E.: Nodal Mixed Discretization for Tetrahedral Elements, in: *Numerical Modeling in Geomechanics—2006: Proceedings of the 4th International FLAC Symposium*, edited by Hart, R. and Varona, P., Itasca Consulting Group, Minneapolis, Minnesota, paper 07-02, Madrid, Spain, May 2006, 2006.
- Duretz, T., De Montserrat, A., Sevilla, R., Räss, L., Utkin, I., and Spang, A.: Automatic Tuning of Iterative Pseudo-Transient Solvers for Modeling the Deformation of Heterogeneous Media, *Geoscientific Model Development*, 19, 5343–5362, <https://doi.org/10.5194/gmd-19-5343-2026>, 2026.

- Hodgkins, W. R.: On the Relation between Dynamic Relaxation and Semi-Iterative Matrix Methods, *Numerische Mathematik*, 9, 446–451, 1967.
- Kelley, C. T. and Keyes, D. E.: Convergence Analysis of Pseudo-Transient Continuation, *SIAM Journal on Numerical Analysis*, 35, 508–523, <https://doi.org/10.1137/S0036142996304796>, 1998.
- Lavier, L. L., Buck, W. R., and Poliakov, A. N. B.: Factors controlling normal fault offset in an ideal brittle layer, *Journal of Geophysical Research: Solid Earth*, 105, 23 431–23 442, <https://doi.org/10.1029/2000JB900108>, 2000.
- Otter, J. R. H.: Computations for Prestressed Concrete Reactor Pressure Vessels Using Dynamic Relaxation, *Nuclear Structural Engineering*, 1, 61–75, 1965.
- Otter, J. R. H., Cassell, A. C., and Hobbs, R. E.: Dynamic Relaxation, *Proceedings of the Institution of Civil Engineers*, 35, 633–656, 1966.
- Poliakov, A. N. B., Cundall, P. A., Podladchikov, Y. Y., and Lyakhovsky, V. A.: An explicit inertial method for the simulation of viscoelastic flow: An evaluation of elastic effects on diapiric flow in two- and three-layers models, in: *Flow and Creep in the Solar Systems: Observations, Modeling and Theory*, edited by Stone, D. B. and Runcorn, S. K., pp. 175–195, Kluwer Academic Publishers, Dordrecht, 1993.
- Räss, L., Licul, A., Herman, F., Podladchikov, Y. Y., and Suckale, J.: Modelling Thermomechanical Ice Deformation Using a GPU-based Implicit Pseudo-Transient Method (FastICE v1.0), *Geoscientific Model Development Discussions*, pp. 1–34, <https://doi.org/10.5194/gmd-2019-249>, 2019.
- Räss, L., Utkin, I., Duretz, T., Omlin, S., and Podladchikov, Y. Y.: Assessing the Robustness and Scalability of the Accelerated Pseudo-Transient Method, *Geoscientific Model Development*, 15, 5757–5786, <https://doi.org/10.5194/gmd-15-5757-2022>, 2022.