

Author reply to Referee #1: egusphere-2026-1180

I thank the referee for a careful and detailed review. The central concern, that some results were framed more broadly than the evidence supported, is accepted. The validation wording, the well-balanced claim and the speedup are narrowed or withdrawn below. In response, the revision adds verification inside a running model, carried to the full length of each test case (RC1-2), a constant-stratification test in which non-zero terms must cancel (RC1-3), and a rebuilt performance section measured against multi-core runs (RC1-4). Section, figure and table numbers refer to the revised manuscript unless stated otherwise.

RC1-1: Verification and validation should be distinguished more clearly

Referee comment:

The study shows that, for identical inputs, the CUDA implementation closely reproduces the existing Fortran routine. It shows that the translation from the CPU implementation to CUDA has been performed correctly and that no obvious numerical or indexing errors remain in the tested configuration. However, this comparison does not independently validate the pressure-gradient scheme itself. Any error already present in the CPU implementation would also be reproduced by the GPU version and would not be revealed by the comparison. I would suggest revising the wording (validation -> verification) in the abstract, Sect. 2.3, Sect. 3, and the conclusions so that the main result is framed as numerical fidelity to the existing ROMS implementation rather than independent confirmation of the scheme's correctness.

Response:

I thank the referee for this distinction, which has been accepted in full. “Validation” has been replaced with “verification” in Sect. 2.3, Sect. 3 and the conclusions, and the corresponding subsections have been retitled. The abstract has been rewritten and uses neither term. It states that the accuracy of the scheme is established in the original literature, and that what is demonstrated here is exact reproduction of the existing Fortran routine. An explicit statement of the limitation now stands at the head of Sect. 2.3:

“What follows is a verification exercise in the sense conventionally distinguished from validation... Close agreement is therefore evidence that the algorithm has been translated to CUDA correctly, but not proof of it... It does not independently confirm the correctness of the Shchepetkin scheme itself. Any error already present in the Fortran implementation would be reproduced faithfully by the GPU implementation and would not be revealed by this comparison.”

The conclusions carry the same qualification.

RC1-2: The seamount experiment is a single-call comparison rather than a dynamic benchmark

Referee comment:

The introduction emphasizes that pressure-gradient errors can generate spurious currents that persist or grow over time. This is also the main reason the seamount problem is widely used as a benchmark. In the present study, however, the comparison

appears to be based on one time step. It does not show how the GPU implementation behaves during a time integration. A stronger experiment would integrate the CUDA kernel within ROMS over many baroclinic time steps and compare diagnostics such as maximum spurious velocity and domain-integrated kinetic energy. If full integration is beyond the scope of the current study, the manuscript should state more explicitly that the present experiment is a single-call verification test and should avoid broader claims about long-term stability or circulation behavior.

Response:

I am grateful for this suggestion. A new Sect. 3.5 reports verification inside a running model. The kernels replace `prsgird32` in the ROMS 4.2 baroclinic loop through an `ISO_C_BINDING` interface, with device buffers kept on the GPU across timesteps. The CPU and GPU executables are built from one source tree and differ only in whether the CUDA path is enabled.

Both diagnostics the referee named, volume-averaged kinetic energy and maximum speed, were compared at every record the model prints, over the full length each case distributes (Table 2):

Case	Run	Ranks	Records	Differing, KE / speed	Final KE ($\text{m}^2 \text{s}^{-2}$)	Final speed (m/s)
Seamount	60 s x 259,200 (180 days)	32	259,201	0 / 7	1.528386e-05	3.980402e-02
Canyon	100 s x 103,680 (120 days)	32	103,681	0 / 0	1.669711e-04	3.921783e-01
Overflow	20 s x 1800 (10 hours)	1	1801	0 / 0	5.205116e-03	1.554973e+00

On Seamount the maximum speed rises from rest to $3.8\text{e-}02$ m/s in the first twenty days and then stays between $3.79\text{e-}02$ and $4.16\text{e-}02$ m/s. The case has no explicit horizontal viscosity, so this bound is set by the quadratic bottom drag and background vertical viscosity as well as by the scheme. Each of the seven differing speed records differs by one unit in the last printed digit, the GPU value larger at four and smaller at three. They show no trend in time, and the kinetic energy agrees at every record. Canyon, whose maximum speed is about ten times larger, has no differing record. The seven are values that fall close to a rounding boundary of the printed number.

Shorter runs were also compared, all at a 0.1 s timestep on Seamount: four grid sizes from $3.1\text{e}4$ to $3.6\text{e}6$ cells, and 1440 steps on the smallest grid. Canyon was also run for 2000 steps at 100 s. The kinetic energy agrees in every printed digit in all of them, and after 1440 steps the velocity fields agree to $1.1\text{e-}9$.

Kinetic energy is a coarse check, so its sensitivity was tested. With the threshold defect of Sect. 3.2 put back into the GPU kernel, the 2000-step Canyon run changes 23 of its 2001 records, first at step 39, in the seventh digit. The live comparison therefore supports the pointwise results, and the bitwise test is the stronger one.

The GPU path was checked directly. The GPU executable contains no `prsgird32` symbol, a run with the GPU hidden stops at startup instead of falling back to Fortran, and during the runs the GPU showed one active compute process per rank.

Under domain decomposition, a 500-step Canyon run gives identical kinetic energy at all 501 records at 1, 2, 4 and 6 ranks, which is consistent with correct use of the halo regions. The full-length runs above used 32 ranks, although tiling invariance itself was checked only up to 6. All ranks share one

GPU, so the routine’s summed cost rises from 0.72 s at one rank to 1.01 s at six, while the CPU build falls from 0.27 s to 0.06 s. Operation with one GPU per rank is not implemented.

The single-call comparison was also extended to two more bathymetries (Table 1, Sect. 3.1):

- Canyon (65x48x15, $r = 0.20$ against 0.22 for Seamount): maximum differences of $7.26\text{e-}7$ and $5.36\text{e-}7 \text{ m}^4 \text{ s}^{-2}$ in ru and rv, or $1.2\text{e-}9$ and $8.8\text{e-}10$ of the peak values, with 45,825 comparable rv points against 28,968 on Seamount.
- Overflow, a gravity-current case whose density varies horizontally: maximum rv difference $1.28\text{e-}7 \text{ m}^4 \text{ s}^{-2}$, or $1.4\text{e-}11$ of the peak, and bit-identical at all 10,136 points under strict arithmetic. It is four cells wide in xi, so ROMS returns zero at every ru point and the kernel does too. The case adds no xi-direction evidence.

Under strict arithmetic, no point in any of the three configurations differs in a single bit. The configurations and reference data are in the archive.

RC1-3: The lake-at-rest test should be described more narrowly

Referee comment:

The current interpretation of the lake-at-rest test appears stronger than the test supports. Because the density anomaly is set to zero everywhere, the density-dependent baroclinic terms vanish directly. The result therefore confirms that the kernel returns zero for a zero-density-anomaly input, but it does not test cancellation among nonzero density, pressure, and geometric terms. I would suggest narrowing the associated well-balanced claim. A more demanding alternative would be to add a case in which the expected tendency is independently known and nonzero intermediate terms must cancel numerically.

Response:

I thank the referee for catching this, and the claim has been narrowed. The conclusions now state that the kernel returns exactly zero pressure gradient force for a vanishing density anomaly, as the CPU routine does, instead of asserting the well-balanced property in general, and Sect. 3.4 adds that on a constant-stratification field neither implementation returns zero. The rewritten abstract makes no lake-at-rest claim at all.

A full-model lake-at-rest experiment is not attainable in this configuration: a uniform-density cold start over the steep Seamount bathymetry excites a barotropic gravity-wave shock that violates the CFL condition before the pressure-gradient routine is reached. That is a constraint of the host model’s time-stepping and not of the scheme.

The test the referee asks for has been added. The density anomaly was replaced by a function of depth alone, $\rho'(z) = \alpha \cdot z$ with $\alpha = 2\text{e-}3 \text{ kg m}^{-4}$, over the unchanged Seamount bathymetry. The density is then horizontally uniform in geopotential coordinates, so the true horizontal pressure gradient is identically zero and the expected answer is known analytically. Along a terrain-following surface, however, the same field varies by 8.16 kg m^{-3} across the deepest sigma level and the kinematic pressure reaches $-2.0\text{e}2 \text{ m}^2 \text{ s}^{-2}$, so the intermediate terms are large and must cancel numerically.

The kernel returns a maximum residual of $1.79\text{e-}4 \text{ m}^4 \text{ s}^{-2}$ in ru, with an rms of $1.01\text{e-}5$, against an exact answer of zero.

For a verification paper the relevant quantity is fidelity to ROMS, not agreement with the analytic limit. The unmodified Fortran routine was therefore applied to the identical input, through a standalone driver that calls the `prsgrd32_tile` body verbatim, and it gives the same values to every digit quoted. Neither implementation returns zero, and the two depart from the analytic limit identically.

The test has one limit. With density linear in depth the two curvature corrections are equal and cancel, so in exact arithmetic the discrete scheme returns zero on this field. What both implementations return is accumulated rounding in the cancellation, not truncation error. The test exercises the cancellation and not the fourth-order corrections, and Sect. 3.4 now says so.

I am grateful for the alternative the referee proposed: it replaced a weak test with one that exercises the cancellation, and narrowed what the paper is entitled to conclude.

RC1-4: The performance comparison should be placed in a more representative context

Referee comment:

The reported 15x speedup is based on comparison with one Intel Xeon Gold 6154 core on a 54x51x13 domain and is not representative of how ROMS is typically used. For practical applications, the relevant CPU runs would involve multiple cores and MPI domain decomposition rather than a single core. The current result should be presented as a preliminary kernel-level comparison for the specific grid and hardware tested. The manuscript should clearly distinguish single-core speedup from node-level or resource-equivalent performance. It would also be useful to report the CPU compiler, optimization flags, and vectorization settings, since the baseline walltime may depend strongly on the build configuration.

The timing methodology also needs to state whether the reported walltime includes memory allocation, host-to-device and device-to-host transfers, kernel launches, synchronization, warm-up iterations, and CUDA context initialization, since the manuscript itself notes that launch and transfer latency may be significant for the small seamount grid.

Response:

I am grateful for this. The performance section, Sect. 3.5 at submission and now Sect. 3.6, has been replaced entirely, is now titled “Computational Cost”, and makes no speedup claim anywhere. Referee #2 raised a closely related point, on sub-domain size and the full socket, and both are answered together here.

The 12 ms single-core figure quoted in the submitted manuscript is not reproducible. The profiler measures 0.674 ms for that routine and grid with ROMS’s default compiler settings, and 0.368 ms once the build is vectorized as described below. Either way the 15x speedup does not stand, and it has been removed.

Methodology. Both builds are timed with ROMS’s own profiler (timer 23). The CPU build already instruments the pressure-gradient routine. The GPU build did not, and the same pair of calls has been added around the identical code region, so the two are measured with the same instrument. ROMS accumulates profiler values across MPI ranks, so the logged quantity is aggregate processor-seconds and mean per-rank time per timestep is obtained by dividing by the

rank count. Build configuration: `gfortran -O3 -ffast-math -march=native -funroll-loops -mprefer-vector-width=512`, and `nvcc -O2` targeting `sm_61` under CUDA 12.4. Timings exclude one-time allocation and CUDA context creation, which are paid once because device buffers persist across timesteps, and include all per-timestep host-device transfer, kernel launches and synchronization. No warm-up iterations are discarded: the profiler accumulates over every timestep of the run, so the first call is included in the average.

Results (mean ms per baroclinic timestep; GPU is one Quadro P1000, CPU a dual Xeon Gold 6154 node with 36 physical cores):

Grid	Cells	GPU	CPU 1 core	CPU 32 cores
49x48x13	30,576	0.943	0.368	0.031
100x100x40	400,000	8.510	6.015	0.360
200x200x40	1,600,000	31.86	31.92	1.514
300x300x40	3,600,000	70.12	77.00	3.786

The correction to the baseline. The referee asks about compiler flags, and checking them revealed that the CPU build behind the first timing sweep made for this revision was compiled with no vectorization: `Compilers/Linux-gfortran.mk` set only `-O3 -ffast-math`, and disassembly confirmed the binary contained no instruction wider than 128 bits. An unvectorized baseline understates the CPU and would have flattered the GPU comparison.

The baseline was rebuilt with the flags above and the entire sweep re-run. The table is the new measurement. The flag `-mprefer-vector-width=512` is necessary: `gfortran` defaults to 256-bit vectors on this architecture to avoid AVX-512 clock throttling, so `-march=native` alone yields AVX2, and AVX-512 instructions were confirmed in the final binary. The routine is 1.78x faster vectorized, the mean of six alternating paired runs of the two builds on the same configuration, with a standard deviation of 0.018. The old binary re-measured at 0.648 ms against the 0.674 ms measured before the rebuild, a difference of under four percent, so the earlier timings were themselves reproducible.

Per device, the implementation is 2.6x slower than a single core on the smallest grid and 1.4x slower at 4e5 cells. It reaches parity at 1.6e6 cells (31.86 ms against 31.92) and is marginally ahead, by 1.10x, at 3.6e6 cells. The 1.06x and 1.29x advantages found in that first sweep do not survive a vectorized baseline.

That is not the operationally relevant comparison, and Sect. 3.6 says so. The Fortran routine strong-scales cleanly, retaining between 37% and 66% parallel efficiency at 32 ranks. Against 32 cores the GPU implementation is 18 to 31 times slower. The fully occupied 36-core node was also timed, in a 6x6 tiling. The routine costs 0.0242, 0.287, 1.289 and 3.175 ms per timestep on the four grids, on average 1.23x faster than at 32 ranks. Against a fully loaded node the GPU is therefore 22 to 39 times slower.

Sect. 3.6 also reports the comparison at realistic per-rank subdomain sizes, 25x50x40 and 38x75x40, where the GPU is 2.3x and 1.7x slower than one core. It identifies the two causes quantitatively. Host-device transfer is 57.8% of the routine’s time, and the kernels achieve about 17% of the device’s memory bandwidth because the thread-local slope arrays are DRAM-backed local memory rather than registers, which is the subject of RC1-5.

RC1-5: The `MAX_N = 64` limitation needs clarification

Referee comment:

The statement that “`MAX_N = 64` accommodates the vertical resolution of any current ROMS configuration” appears too broad. Please clarify whether `MAX_N` is a compile-time parameter that users can modify, what happens when $N > 64$, and whether the code fails with a clear error.

Response:

I thank the referee for pressing on this. The statement was too broad and the code did not fail cleanly. Both are fixed.

`MAX_N` is a compile-time constant bounding the thread-local slope arrays. Users may raise it and recompile (`-DMAX_N=128`). The released code now checks N against `MAX_N` before launch and aborts with an explicit message naming the required rebuild, rather than writing past the arrays.

Examining the bound corrected a factual error in Sect. 2.2, which stated that the arrays are held in registers. They are not. The compiler reports, for `sm_61`, 56 registers per thread and a stack frame of $32 * \text{MAX_N} + 16$ bytes, 2064 bytes at `MAX_N = 64`, with no spilling, and the register count is independent of `MAX_N`. Because the vertical loop bounds are runtime values, the compiler cannot unroll the loops and promote the arrays, so it places them in per-thread local memory. The cost of raising `MAX_N` is therefore local-memory footprint and occupancy, not register pressure. Sect. 2.2 now states this with the measured figures, and Sect. 3.6 shows it is the dominant kernel inefficiency.

RC1-6: Figure 2a color bar

Referee comment:

Figure 2a is described as showing absolute error, but the color bar contains both positive and negative values. Please revise either the plotted quantity or the caption and color-bar label.

Response:

I am grateful for this catch, and the panel has been corrected. The referee’s Figure 2 is still Figure 2. The flowchart formerly shown as Figure 1 has moved to the Supplement, and two figures have been added: Figure 1 gives the bathymetry of the three test configurations, and Figure 3 the maximum difference at each stage of this work. Panel (a) plotted the signed difference with a diverging color map while being described as an absolute error. It now plots $|\text{ru_GPU} - \text{ru_CPU}|$ on a single-hue sequential map with a logarithmic norm, and the plotted quantity, the color-bar label and the caption agree.

Revising the figure exposed two further problems, both corrected.

The caption and Sect. 3.2 of the submitted version stated that differences are largest at the domain boundaries. They are not. The ROMS routine does not evaluate `ru` on the outermost ring, so the reference is identically zero there, while the standalone kernel, launched over the whole array, does evaluate those points. The apparent difference is the kernel’s own output compared against zeros. Restricted to every point at which ROMS returns a non-zero value, the maximum difference over the entire domain is $4.22764 \times 10^{-6} \text{ m}^4 \text{ s}^{-2}$, identical to the interior maximum and located at an

interior point on the steep flank. Points at which ROMS does not evaluate ru are now masked in the figure and the text explains why. In the integrated configuration the question does not arise, since the kernels are launched over the model’s own tile bounds with halos populated by ROMS.

Panel (c) was described as showing uniform $O(10^{-3})$ accuracy with no error accumulation from surface to seabed. The RMS difference varies strongly with depth, tracking ru itself. With the corrected kernel it runs from $1.05e-11 \text{ m}^4 \text{ s}^{-2}$ at the surface to $1.05e-6$ at the seabed, against an ru spanning $9.1e-3$ to $7.2e3$ over the same levels. Normalized by the local signal, the agreement is a near-uniform $O(10^{-10})$. The caption now states this and the axis is logarithmic. The submitted Sect. 3.2 carried the same incorrect description, and also described the distribution of differences as approximately Gaussian. The revised Sect. 3.1 gives its actual shape: mean $2.5e-11 \text{ m}^4 \text{ s}^{-2}$, standard deviation $4.0e-7$, excess kurtosis 15. Before the threshold defect described at the end of this reply was corrected, the excess kurtosis was of order $4e2$. That value was the signature of the defect, a small subset of points behaving differently, which is what a diverging conditional branch produces and accumulated rounding does not.

RC1-7: Consistency of the “first publicly available” claim

Referee comment:

The “first publicly available” claim is stated definitively in the abstract and conclusions but qualified as “to our knowledge” in Sect. 1.2. Please use consistent wording throughout and briefly indicate the basis for the claim.

Response:

I thank the referee for asking for consistency here. The qualified form, “to our knowledge”, has been adopted, and the basis stated: a literature search for GPU implementations of ROMS and of the density-Jacobian scheme, inspection of the public ROMS repository and its principal forks, and review of the GPU-capable ocean models now discussed in Sect. 1.2, none of which implement this scheme. The text notes that an unpublished or internal implementation cannot be ruled out.

The claim now appears in Sect. 1.2, again in Sect. 4.1, and once more in the conclusions, in the same qualified form each time. It is no longer in the abstract, which goes beyond what this comment asked for. The submitted abstract presented the work through the priority claim. The revised abstract states the verification result of Sect. 3.2 in its body and describes the CUDA implementation instead of announcing it. The priority claim is not withdrawn.

Changes that affect numbers in the submitted manuscript

Two defects in the kernel, found by following the other referee’s suggestion of a bitwise comparison. (Sect. 3.3 of the submitted version, which discussed the earlier indexing error, is Sect. 3.2 of the revised manuscript; Sect. 3.3 is now land masking.) That suggestion prompted a line-by-line audit of the CUDA sources against `prsgird32.h`. It found a mismatch in the horizontal harmonic-mean slope reconstruction: ROMS tests `cff1 = 2.0_r8*FC(i,j)*FC(i+1,j)` against `eps`, while the kernel tested the bare product. Its threshold was therefore twice as strict, and it returned zero where the Fortran computes a slope. The affected code was in the archived deposit. With the defect corrected, the maximum difference in ru falls from $4.46e-2$ to $4.22764e-6$. Every compared point now lies within the harness tolerance, an absolute criterion of $1e-5 \text{ m}^4 \text{ s}^{-2}$ defined in Sect. 2.3.3; previously 852 points did not. That tolerance is weak, the maximum difference sitting only a factor of 2.4 below it, which is why the paper rests on the bitwise result instead.

The bitwise test itself then found a second defect that the audit had passed. The horizontal harmonic mean divided directly where ROMS forms a reciprocal and multiplies, a difference of one unit in the last place. A line-by-line reading did not catch it, since both forms compute the same quantity. It affected two of the 85,501 compared points on the Canyon configuration, each by $2.9 \times 10^{-9} \text{ m}^4 \text{ s}^{-2}$, a size no tolerance-based criterion would have flagged.

With both corrected, agreement under strict arithmetic is exact on all three configurations; RC1-2 reports the result. The correction changes the agreement figures throughout, and Figure 2, which is the subject of RC1-6. The submitted manuscript presented the $O(10^{-6})$ residual as intrinsic to floating-point arithmetic on differing hardware. It was dominated by the threshold defect.

Land masking. At submission the kernels did not implement the MASKING option of `prsgd32.h`. Almost every realistic coastal configuration defines it, and without it the kernel reconstructs density slopes straight through land. Masking has been implemented and verified bit for bit against the Fortran body compiled with `-DMASKING`, on a domain with an island carved out of it. A 1440-step masked model run corroborates this independently, with kinetic energy identical and the fields agreeing to 1×10^{-9} relative. A new Sect. 3.3 reports both. It includes the negative control: with masking disabled and the masked reference retained, the maxima are 2.4×10^3 and $4.9 \times 10^4 \text{ m}^4 \text{ s}^{-2}$ at 4,539 and 5,345 points. On a domain with land, an unmasked kernel gives no indication of the error.

The revised contribution list

Sect. 1.3 now states five contributions instead of three. The two added are verification inside a running model and the verification sequence itself. Two of the original three are narrowed: the second no longer relies on the lake at rest, which Sect. 3.4 shows to be a weaker test than it appears, and the third no longer claims that other terrain-following models can adopt the kernels unchanged. The paper’s overall claim is now fidelity, the demonstration that the complete scheme can be reproduced exactly.

A correction to the single-precision estimate

Sect. 2.2.2 previously stated that single precision would introduce $O(10^{-7})$ relative errors. Direct measurement gives a very different figure. Recompiling the kernel in single precision and applying it to the Seamount case produces a maximum `ru` error of $1.2 \times 10^3 \text{ m}^4 \text{ s}^{-2}$ against a $\pm 7200 \text{ m}^4 \text{ s}^{-2}$ reference field, that is, approximately 17% of that field’s peak magnitude. The cause is that the kinematic pressure accumulates to $O(10^3) \text{ m}^2 \text{ s}^{-2}$ through the depth integral of the density anomaly over a 5000 m water column, while the gradient is formed from differences of such quantities. Seven significant digits cannot preserve that cancellation. The text now reports the measured value, which shows that double precision is a requirement of the scheme.

A correction to the units of `ru` and `rv`

The submitted manuscript gave `ru` and `rv` in $\text{m}^2 \text{ s}^{-2}$, which is the unit of the kinematic pressure and not of the momentum forcing terms. ROMS forms `ru = on_u * 0.5*(Hz+Hz) * (P(i-1)-P(i)-...)`, a grid spacing times a layer thickness times a kinematic pressure, so the unit is $\text{m}^4 \text{ s}^{-2}$. No measured value changes and no result or conclusion is affected. The unit is corrected throughout the revised manuscript, the Supplement and this reply, and Sect. 2.1.3 now states the dimension where `ru` is defined.

I am grateful to the referee for comments that led directly to the live-model, well-balanced and performance work in this revision.