

Author reply to Referee #2 (Andrew Porter): egusphere-2026-1180

I thank Dr Porter for a detailed and constructive review. Both major concerns are answered with new work: the kernels now run inside ROMS 4.2 (RC2-1), and the same scheme has been implemented with OpenACC directives for comparison (RC2-2). The referee's suggestion to seek bitwise reproducibility became the central result of the revision (RC2-3). Section, figure and table numbers refer to the revised manuscript unless stated otherwise.

RC2-1: The port tackles only one component of the model

Referee comment:

My primary concern with this work is that it tackles only one component of the ROMS ocean model. To use this in production as provided would mean paying the price of moving data between CPU- and GPU-local memory spaces every time the routine is called. This would significantly impact any performance advantage obtained through use of the GPU.

Response:

I thank the referee for this. The prediction is correct, and the revision measures it.

Host-device transfer accounts for 57.8% of the routine's time on a 200x200x40 grid and 62.6% on the smaller Seamount grid. As the referee anticipated, the GPU routine has no performance advantage at this scope. Against one vectorized CPU core the routine is 2.6x slower on the smallest grid and reaches parity only at 1.6e6 cells. At the per-rank subdomain sizes a 32-rank production run presents, it is 1.7 to 2.3 times slower than a single core. Against 32 cores of the same node it is 18 to 31 times slower, and the crossover is at one to two cores. In the full-length runs of Sect. 3.5 all 32 ranks share one device, and the whole model takes about four and a half times as long as the CPU build. Sect. 3.6, Table 3 and Table 4 report this.

The cost is intrinsic to accelerating a single routine within an otherwise host-resident model. The fields the kernel consumes are produced on the host each timestep and must be moved, and kernel optimization does not reduce this component. The transfers use pageable host memory over PCIe, and page-locked allocations would be expected to improve the rate but not the conclusion. Sect. 3.6 identifies the next step, moving neighboring routines to the device so that the model state becomes resident.

The paper therefore claims fidelity. The speedup figure reported in the submitted version has been withdrawn.

The port is also no longer a standalone kernel. It has been integrated into ROMS 4.2 and run inside the model. The kernels are called through an ISO_C_BINDING interface that replaces `prsgird32` in the baroclinic time-stepping loop, with the Fortran call site otherwise unmodified. Two executables were built from an identical source tree, differing only in whether the CUDA path is enabled, so any difference between them is attributable to the pressure gradient alone. A new Sect. 3.5 reports live-model agreement on all three configurations, Seamount, Canyon and Overflow, each carried to the full length its case distributes: 180 days on Seamount, 120 on Canyon, and 10 hours on Overflow. Across 259,201 and 103,681 reported records the two solutions agree in volume-averaged kinetic energy in every digit the model reports.

The kernels were also run under the model's own domain decomposition, at 1, 2, 4 and 6 MPI

ranks over 500 timesteps. Kinetic energy is identical at all 501 reported records, at every rank count and against the CPU build. That is consistent with correct consumption of the halo regions. It is a correctness result and not a scaling one. The full-length runs were made at 32 ranks on both builds, although tiling invariance itself was checked only from 1 to 6 ranks. All ranks share the single device available, and operation with one device per rank is not implemented (Sect. 4.5). The routine's cost summed across ranks rises from 0.72 s at one rank to 1.01 s at six as they serialize on it, while the CPU build falls from 0.27 s to 0.06 s.

Volume-averaged kinetic energy is a coarse diagnostic, so its sensitivity was calibrated. Reintroducing the threshold error of Sect. 3.2 into the GPU kernel alone and repeating the Canyon integration alters 23 of the 2001 records, the first at step 39, and leaves the final value unchanged. The diagnostic detects a defect of that size, but only in the seventh reported digit and at a small fraction of steps. The live agreement therefore corroborates the pointwise results, and the bitwise comparison is the stronger test.

RC2-2: The choice of the proprietary CUDA language

Referee comment:

My second major concern is the choice of the proprietary (NVIDIA) CUDA language for the port. This means that the new routine can only be compiled with the NVIDIA compiler and will only run on NVIDIA hardware. Maintainability of the port is not discussed and this is likely to be an issue for model developers who are often used to working in Fortran and Python. An implementation using OpenACC or OpenMP directives would retain the Fortran base, be portable across GPU vendors and would be a valuable comparison point in terms of code complexity and computational performance.

Response:

I am grateful for this, and both criticisms stand. Maintainability had not been discussed at all. A new Sect. 4.4, "Portability and maintainability", states that CUDA C restricts the routine to NVIDIA hardware and the NVIDIA toolchain, and introduces a second language into a codebase maintained by a Fortran- and Python-centered community. Where the goal is to move an existing model to accelerators with least disruption to its developers, directives are the more sustainable route.

The vertical integration is a sequential recurrence in k . The decomposition that suits it parallelizes over the horizontal indices (i,j) , keeps k sequential within each thread, and holds the slope arrays in thread-local storage. That is not the mapping a compiler selects unaided from the loop nest. An explicit implementation therefore specifies what a directive-based port must achieve, structurally and numerically, and supplies a characterized performance reference for the same algorithm on the same hardware. Sect. 4.4 also describes the maintenance surface. The `ISO_C_BINDING` boundary passes array base addresses and dimensions, the Fortran call site is otherwise unmodified, and the CUDA sources have no dependency on ROMS data structures. The kernels can therefore be replaced behind that interface by a directive-based version, a HIP translation or anything else.

The comparison the referee asked for has been carried out. The same scheme was implemented in Fortran with OpenACC directives and compiled with `nvfortran` 24.5 for the same Quadro P1000. It was compared against both the ROMS reference and the CUDA kernels in the standalone binary-extraction harness, which needs no ROMS build. The method and full results are in the Supplement.

	CUDA	OpenACC
max abs diff vs ROMS, 54x51x13	4.227644e-6	4.227644e-6
max abs diff vs ROMS, 206x203x40	4.765892e-7	4.765892e-7
kernel time, 206x203x40 (median of 6 interleaved runs, 100 reps each)	11.90 ms	12.01 ms
non-comment lines, total	273	147
of which compute kernels	155	144
of which memory management	89	3 directives
of which includes, macros, boilerplate	29	none

The two implementations produce bit-for-bit identical output at both grid sizes, and so depart from the Fortran reference by exactly the same amount, despite different languages and different compilers. This supports Sect. 4.2: the CPU-GPU difference comes from how the compiler selects instructions, in particular fused multiply-add contraction, which both GPU implementations share. It is not specific to CUDA. Reordered summation plays no measurable part, since the vertical recurrence executes in the same sequence within a thread as it does within the Fortran loop. The directive-based version has also been run directly against the strict-arithmetic Fortran reference, so the result no longer depends on the CUDA implementation. Built with `-gpu=cc61,nofma`, it reproduces `prsgird32.h` bit for bit at the same 29,952 ru and 29,440 rv points, with no differing bit pattern. Sect. 4.4 now reports that measurement in place of the inference.

On performance, no ordering between the two is claimed. The paired difference is 0.070 ms, or 0.6%, over six interleaved pairs of runs on the 206x203x40 grid, each run averaging 100 kernel invocations; the medians are 11.90 ms and 12.01 ms. Interleaved pairs were used because shorter runs on this device were disturbed by other activity: earlier sessions gave differences of 0.8% one way and 9% the other. The small Seamount grid is not informative, because at 2,754 columns launch overhead dominates. The CUDA version is therefore no faster than the OpenACC version for this scheme on this hardware, and its value is as a verification reference. The code-size advantage is smaller than the totals suggest. The kernels themselves are of comparable length, 144 lines against 155, because they express the same algorithm. The saving is in what surrounds them: 89 lines of explicit allocation, transfer and launch management, which three directives replace.

The OpenACC version was compiled and run only with `nvfortran` on the NVIDIA device used throughout, so what is shown is portability of the source and not of the result; the revised Sect. 4.4 says so. The decomposition still had to be specified rather than discovered. `-Minfo=accel` confirms that `nvfortran` generated `gang, vector(128) collapse(2)` over `(i,j)` with `loop seq` on each of the three vertical loops, which is the mapping of Sect. 2.2. A `collapse(3)` over the nest would be incorrect because the vertical integration is a recurrence, and the compiler has to be told this. `-Minfo` also reports local memory used for the private slope arrays, as `ptxas` does for CUDA. The bandwidth limitation of Sect. 3.6 therefore follows from the decomposition, not the language, and the directive port inherits it.

The directive version was written after the verified CUDA implementation and guided by it, including the harmonic-mean indexing that the CUDA port initially got wrong. That is the

reference-implementation role described in Sect. 4.4, and not an independent reimplementation.

The conclusion has moved toward the referee's: for this kernel a directive-based implementation is competitive on performance, smaller, portable in source, and maintainable in Fortran (Sect. 4.4).

RC2-3: Compiler flags and bit-wise reproducibility

Referee comment:

The paper does do a thorough job of investigating the differences in results seen on the CPU and GPU. This would benefit from consideration of compiler flags too. We have found that it is often possible to get bit-wise reproducibility between CPU and GPU when appropriate flags are used. This provides an extra level of confidence that the implementation is correct (but can be relaxed under scientific guidance).

Response:

I am grateful for this suggestion. Following it led to the bitwise result of Sect. 4.2 and exposed two defects in the kernel, one by the source audit it prompted and one by the bitwise test itself.

First, the defects, because the corrected numbers depend on them. Auditing the CUDA sources line by line against `prsgrd32.h`, instead of continuing to compare outputs alone, found a mismatch in the horizontal harmonic-mean slope reconstruction. ROMS forms `cff1 = 2.0_r8*FC(i,j)*FC(i+1,j)` and tests that quantity against `eps`. The kernel tested the product `FC(i,j)*FC(i+1,j)` instead, so its threshold was twice as strict and it returned zero at points where the Fortran computes a slope. The Seamount configuration shows where such points lie. Its density is horizontally uniform by construction, so the horizontal density differences along a sigma surface come from the coordinate slope alone. They vanish over the flat plain and grow across the flank, so a ring of points between the two falls in the affected band. The submitted manuscript's $O(10^{-6})$ agreement, and its interpretation of that agreement, both followed from this defect. The affected code was in the archived deposit.

With the threshold corrected, the agreement is four orders of magnitude closer, and every compared point falls within the harness tolerance, an absolute criterion of $1e-5 \text{ m}^4 \text{ s}^{-2}$ now stated in Sect. 2.3.3, where previously 852 points did not:

	before correction	after correction
Seamount max diff, ru	4.45841e-2	4.22764e-6
Seamount max diff, rv	3.81999e-2	4.13943e-6
points within tolerance, ru	28,156 / 29,008	29,008 / 29,008
Canyon, ru / rv	1.02e-2 / 8.9e-3	7.26e-7 / 5.36e-7
Seamount peak-normalized, ru	6.19e-6	5.87e-10

Second, bitwise agreement is attainable, and it has been obtained. The CPU reference was regenerated from a ROMS build compiled with `-O2 -fno-fast-math -ffp-contract=off -fno-unsafe-math-optimizations`, and the kernels were rebuilt with `nvcc --fmad=false`. Against that reference the CUDA implementation reproduces the Fortran routine bit for bit on the Seamount case. Across the 29,952 ru points and 29,440 rv points at which `prsgrd32` returns a non-zero value, no point differs in a single bit of its IEEE-754 representation. This was verified independently of the test harness by comparing raw bit patterns.

Repeating the test on Canyon exposed a second defect. The agreement there was at first nearly exact: all 39,676 ru points bit-identical, and 45,823 of the 45,825 rv points, with two differing by $2.9\text{e-}9 \text{ m}^4 \text{ s}^{-2}$ against local values of -29.4, a relative difference of $1\text{e-}10$. That is two points in 85,501, at a relative size indistinguishable from rounding noise. Any tolerance-based criterion would have passed them. Under the bitwise criterion they could not be ignored, and tracing the pair identified a second departure from `prsg32.h`. The Fortran does not divide when it forms the horizontal harmonic mean: it computes `cff2 = 1.0_r8/(FC(i,j)+FC(i+1,j))` and returns `cff1*cff2`. In the kernel the quotient was evaluated directly. The two are algebraically identical but can differ by one unit in the last place: a correctly rounded division commits one rounding error, while a reciprocal followed by a multiplication commits two. The vertical block of the same routine does divide, so the two parts must be transcribed differently to match, and the port had written both the same way. The discrepancy surfaced in rv and not in ru at the same point because of a cancellation there: the eta-direction difference $P(i,j-1)-P(i,j)-J$ is $-7.3\text{e-}5$, formed from terms of order $1\text{e}2$, about six decimal digits of cancellation. That lifts a one-ulp perturbation into the eleventh significant figure, consistent with the relative difference of $1\text{e-}10$.

With the harmonic mean in the reciprocal-multiply form, agreement is exact on both configurations: 29,952 ru and 29,440 rv points on Seamount, 39,676 ru and 45,825 rv points on Canyon, not one differing bit. Under default compiler settings the correction is invisible: every figure in Table 1 is unchanged to the last digit quoted, and the live-model results of Sect. 3.5 are unchanged. This second defect survived the structural audit that had found the first, and only a comparison without tolerance could reach it.

The bitwise test has since been extended to a third configuration, the `OVERFLOW` case, a gravity-current test in which a dense plume descends a slope. The agreement there is likewise exact, 10,136 rv points with no differing bits, and the tolerance-level difference is the smallest of the three at $1.4\text{e-}11$ of the peak signal. It is a vertical slice four cells wide in xi, so the Fortran returns exactly zero at every ru point of its computational range. The kernel returns exactly zero at those points, which is correct, but the case carries no xi-direction evidence. As a control, the strict-arithmetic kernels were run against the default-flags reference for the same case, giving 3,512 differing bit patterns, confirming that the exact agreement reflects the matched build and not a comparison that always succeeds.

All of this holds with the thread-per-column decomposition unchanged. The submitted manuscript presented the $O(10^{-6})$ difference as intrinsic to IEEE-754 arithmetic on differing hardware, and as the behavior expected of one code base across compiler versions and decompositions. That reading was wrong. The residual was dominated by a defect.

Third, fused multiply-add is the only remaining mechanism. With the defects removed, disabling contraction changes the maximum ru difference from $4.23\text{e-}6$ to $1.52\text{e-}7$ against the original reference, and to exactly zero against the strict-arithmetic reference. The submitted manuscript named contraction as one of three mechanisms, alongside non-associative summation and amplification by cancellation, and separated none of them. Against a residual dominated by the threshold defect, its contribution could not be seen.

A methodological note. The first implementation defect in this work, an off-by-one in the vertical harmonic mean, was caught by point-by-point comparison against the CPU reference. The threshold defect was not, and survived that comparison throughout development. It survived because its residual looked like ordinary floating-point differences between two implementations, in both size and spatial pattern. The symptom was in fact present in statistics already being computed:

an excess kurtosis of order $4e2$, with a small minority of points carrying nearly all the discrepancy. That is the signature of a conditional branch diverging on a subset of the domain, not of accumulated rounding. The submitted manuscript described the same distribution as approximately Gaussian. Correcting the threshold drops the excess kurtosis to 15.

Sect. 3.2, formerly Sect. 3.3, has been rewritten around this contrast. Comparing outputs establishes that two implementations behave alike within a tolerance, but cannot establish that they are the same algorithm, since any discrepancy small enough to resemble arithmetic noise will be ascribed to it. A structural comparison of the sources settles that question, and a bitwise test settles it mechanically.

Sect. 4.2 has been rewritten around the result. A GPU port of this scheme can be pinned to the CPU routine bit for bit by building both sides under strict arithmetic, with contraction prohibited in both compilers. This is a stricter optimization regime on each side, with no change to the algorithm, the decomposition or the data layout, and it makes an exact regression test available.

RC2-4: Related work: other GPU ocean models

Referee comment:

In terms of related work, only the port of the Princeton Ocean Model is cited. It would also be interesting to know how this work compares with what is done in other ocean models that run on GPU e.g. Oceananigans and Veros.

Response:

I thank the referee for these references. Sect. 1.2 now discusses Oceananigans (Ramadhan et al., 2020) and Veros (Häfner et al., 2018) alongside the existing POM.gpu reference (Xu et al., 2015), and adds the ROMS GPU work of Mak et al. (2011) and Panzer et al. (2013). Oceananigans and Veros were designed for accelerators from the outset, in modern languages, and can choose discretizations, memory layouts and data structures to suit the hardware.

That freedom is unavailable to an established model. Porting ROMS is a reproduction problem rather than a design problem. The existing scheme must be preserved exactly, and success is measured by agreement with the original. The revised text makes this contrast explicit.

Veros also serves as a calibration point. Häfner et al. report Veros running at approximately half the speed of the MPI-parallelized Fortran code it derives from on 24 CPU cores, reaching parity only on a high-end GPU, for a model written for accelerators throughout. Large speedups should not be expected from porting individual routines of an otherwise host-resident model, and Sect. 1.2 says so with a forward reference to Sect. 3.6.

Sect. 1.2 has also been made more precise about what POM.gpu ported. Xu et al. note that mpiPOM offers a fourth-order internal pressure gradient option, without stating that POM.gpu ports it. Berntsen and Oey (2010) assess second-, fourth- and sixth-order variants of the POM integral approach and favor the fourth-order method of McCalpin (1994), whose stencil adds two points to the second-order scheme. Stencil evaluations of that kind carry over to GPU data parallelism. The density Jacobian does not. It is a sequential recurrence down each column with data-dependent branches in the harmonic mean. Berntsen and Oey note that tests on the signs of local derivatives may reduce computational efficiency, particularly on vector processors. That is the structural reason this routine resists both automatic parallelization and a directive-based collapse, and why the mapping was written out explicitly.

Sect. 1.2 now states a more basic difference. The error this scheme exists to control is specific to terrain-following coordinates, where the coordinate surfaces are not level and the horizontal pressure gradient emerges as a small residual between two large terms. Neither comparison model is terrain-following. Oceananigans is formulated on Cartesian and curvilinear grids, and Veros inherits the geopotential vertical coordinate of the code it derives from. The problem addressed here does not arise for either in the form it takes in ROMS, and neither, to my knowledge, implements the Shchepetkin density-Jacobian scheme.

RC2-5: Computational performance: sub-domain size and the full socket

Referee comment:

Finally, the paper includes very little detail on computational performance: comparing a complete GPU to a single CPU core is not sufficient. A proper comparison needs to consider the typical size of a sub-domain within an MPI decomposition of the full model domain and employ the full CPU socket.

Response:

I appreciate this correction to the comparison’s scope. Comparing a whole GPU with a single CPU core is not a meaningful measure of practical benefit. 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. The 12 ms single-core figure quoted at submission is not reproducible: the profiler measures 0.674 ms for that routine and grid with ROMS’s default compiler settings.

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 only at startup because device buffers persist, and include all per-timestep 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 (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

A correction to the CPU baseline. Checking the compiler flags, as Referee #1 asked, showed that the CPU build behind the first timing sweep made for this revision had 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, so the error would have flattered the GPU comparison had it reached the revised manuscript.

The baseline was rebuilt with the flags above and the entire sweep re-run. The table is the new measurement. Vectorization needs `-mprefer-vector-width=512`, because 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. Re-measured, the old binary gives 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.

With the whole grid on one GPU, 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. ROMS is run across all available cores. The Fortran routine strong-scales cleanly, retaining between 37% and 66% parallel efficiency at 32 ranks, broadly rising with grid size, though not monotonically. Against 32 cores the GPU implementation is 18 to 31 times slower. The referee asks specifically for the full socket. This node has two 18-core sockets, and the comparison is made against both. Because the sweep doubles the rank count at each step, that table stops at 32; 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, so against a fully loaded node the GPU is 22 to 39 times slower rather than 18 to 31 times. The conclusion is unchanged and slightly strengthened.

The per-rank subdomain comparison. The figures above give one device an entire grid. In a decomposed run each rank owns a subdomain, and a multi-device configuration would give each device only that. Both implementations were therefore timed on the subdomains a rank owns at 32 ranks, 25x50x40 and 38x75x40. There the GPU takes 1.74 and 3.12 ms against 0.745 and 1.88 ms for one core, that is, 2.3x and 1.7x slower. The parity seen on whole grids does not survive decomposition, because at per-rank working-set sizes the device is too lightly loaded to cover its per-call costs. On this evidence extending the single-device integration to multiple GPUs would not be worthwhile at this scope, absent the residency changes identified as the next step. Sect. 3.6 states the tiling used, 8x4 at 32 ranks, and Table S4 of the Supplement the resulting per-rank subdomain at each grid, approximately 6x12, 13x25, 25x50 and 38x75 columns. The smallest case is near the size at which decomposition ceases to be meaningful.

Why. Sect. 3.6 identifies both causes. Host-device transfer dominates, as set out under RC2-1, and the kernels reach about 17% of the device’s memory bandwidth because the thread-local slope arrays are DRAM-backed local memory rather than registers. Both are addressable, and neither arises from the scheme’s arithmetic or bears on the port’s fidelity. Sect. 3.6 and Sect. 4.5 name two optimizations, moving neighboring routines to the device and replacing the slope arrays with a rolling two-level register window, which would also remove the MAX_N bound. Neither has been attempted. The implementation reported and verified here is the one deposited in the archive, and either change would require the verification to be repeated in full.

These results are not projected onto data-center accelerators. The dominant cost is transfer, which is a property of the interconnect and of the single-routine scope, not of the device. A faster accelerator would reduce kernel time but not transfer time, and would not alter the conclusion at

this scope.

Land masking

At submission the kernels did not implement the `MASKING` option of `prsg32.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 $1e-9$ relative. A new Sect. 3.3 reports both, including the negative control: with masking disabled and the masked reference retained, the maxima are $2.4e3 \text{ m}^4 \text{ s}^{-2}$ in `ru` and $4.9e4$ in `rv`, with 4,539 and 5,345 points differing. Nothing in the model reports this; an unmasked build runs to completion.

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.

A note on the abstract

The abstract has been rewritten. The submitted version presented the implementation as the first publicly available one and enumerated results. The revised version states what is presented: a high-fidelity implementation of the complete scheme in CUDA C, a directive-based Fortran counterpart for comparison, and the tests needed to establish that such a port is faithful. It carries no references and no undefined abbreviations, in line with the journal's guidance. Every claim in it is made in the body of the paper.

On the significance of the contribution

Since the review, the paper's central claim has changed, and so has the evidence behind it. The review grants that the scheme has not previously been ported to GPU, and the revision is about how faithfully this port reproduces the original and how that can be checked. What is new:

- bit-for-bit agreement with `prsg32.h` under strict arithmetic on three bathymetries, and with land masking active;
- the three defects of Sect. 3.2, two of which passed every tolerance-based comparison applied to them;
- live-model verification, now at the full length each test case distributes: 180 days on Seamount, 120 on Canyon and 10 hours on Overflow, with 259,201 and 103,681 reported records on the first two;
- a second implementation of the same scheme in Fortran with OpenACC directives, bit-identical and matching it in speed, reported in full although it does not favor the approach taken here;
- a deposit that reproduces the central result from pre-extracted state with one command and no ROMS build, including a negative control that demonstrates the comparison can fail.

The 15x speedup of the submitted manuscript appeared three times, each time qualified as preliminary, and was not in the abstract. It is withdrawn nonetheless (RC2-5), and I am grateful to the referee for pressing on the performance reporting.

The contribution is fidelity, tested over the full length each case specifies. Over 180 days on Seamount the kinetic energy is identical at all 259,201 records. The maximum speed differs at seven of them, each time by one unit in the last printed digit. After the first twenty days it stays between $3.79\text{e-}02$ and $4.16\text{e-}02$ m/s, a bound set by the case's bottom drag and vertical viscosity as well as by the scheme. Canyon, whose maximum speed is about ten times larger, has no differing record in 120 days, so the seven Seamount records reflect rounding of the printed value.

Panzer et al. (2013), extending the CUDA Fortran implementation of Mak et al. (2011), profiled ROMS, found the barotropic step at 53 percent of runtime and the pressure gradient at 3 percent, and ported the former while leaving the pressure gradient in Fortran. That choice suits a throughput goal. The pressure gradient, however, carries the discretization error specific to terrain-following coordinates, so its share of runtime is a poor guide to its effect on the model.

A center running ROMS operationally cannot adopt a component that changes its answers, and agreement within a tolerance does not show that a component leaves them unchanged. In this work two of the three defects passed every tolerance-based comparison. A bitwise comparison against the original, applied to more than one bathymetry, would have exposed both. The same question arises for directive-based transformation and automated code generation of Fortran geoscience codes, and this routine, with its recurrence and data-dependent branches, is a hard case for them.

The archive offers two things. The first is a reference implementation of the complete scheme, characterized numerically and in cost on a named device. The second is the verification procedure of Sect. 3.2, with its harnesses and extracted reference states for three bathymetries and the masked case. Both can be used to check a port written in any language.

The paper is offered as a development and technical contribution, not as a performance result.

I am grateful to Dr Porter for a review that changed what this paper claims.