https://doi.org/10.5194/egusphere-2026-1180
Preprint. Discussion started: 22 June 2026

A High-Fidelity CUDA Implementation of the Shchepetkin Density-Jacobian Pressure Gradient Scheme for ROMS

Pavel Alam Munshi[1]

[1]Institute of Oceanography, National Taiwan University, No. 1, Section 4, Roosevelt Rd, Da'an District, Taipei City, 106, Taiwan

Correspondence to: Pavel Alam Munshi (ocean@pavel.science)

Abstract. Baroclinic pressure gradients in terrain-following ocean models require careful numerical treatment to avoid generating spurious currents over steep topography. The Shchepetkin density Jacobian algorithm achieves high accuracy through harmonic mean density slope reconstruction and fourth-order monotonized cubic polynomial corrections, but has largely remained on CPU architectures despite growing GPU adoption in Earth system models. This work presents the first publicly available CUDA implementation of the complete Shchepetkin density-Jacobian pressure gradient scheme as used in the Regional Ocean Modeling System (ROMS). No algorithmic simplifications were introduced during GPU porting. Validation against CPU reference solutions on the "Tall Isolated Seamount" benchmark (Beckmann and Haidvogel, 1993) with steep topography ($r \approx 0.4$) demonstrates maximum relative error of $\mathcal{O}(10^{-6})$ across the $54 \times 51 \times 13$ computational domain. A "lake at rest" test using a horizontally uniform density field over the same Seamount bathymetry confirms the well-balanced property: the kernel produces exactly zero pressure gradient force to machine precision. The small discrepancies in the Seamount comparison reflect floating-point precision and architectural differences between GPU and CPU rather than algorithmic deficiencies. The standalone kernel architecture enables integration into existing ocean models. Source code and validation data are made publicly available under an open-source license.

1 Introduction

1.1 Pressure Gradient Errors in Terrain-Following Coordinates

Ocean circulation models employing terrain-following vertical coordinates face a fundamental computational challenge in accurately computing the baroclinic pressure gradient force over steep topography (Haney, 1991; Mellor et al., 1998). The difficulty arises from a cancellation problem inherent to the sigma-coordinate transformation: horizontal pressure gradients along sigma surfaces contain a large term proportional to the coordinate slope that must cancel nearly perfectly against the hydrostatic pressure gradient. Individually, each term is $\mathcal{O}(10^4)$ Pa m^{-1}, while their physical difference—the baroclinic pressure gradient driving ocean circulation—is typically $\mathcal{O}(10^{-2})$ Pa m^{-1}. A relative truncation error of 0.01% in either term therefore produces an error on the order of the signal itself, generating what has become known as the spurious pressure gradient error. Over steep bathymetry, these errors can drive artificial circulation patterns that overwhelm the physical dynamics.

https://doi.org/10.5194/egusphere-2026-1180
Preprint. Discussion started: 22 June 2026

The "Seamount Problem" described by Haney (1991) and formalized by Beckmann and Haidvogel (1993) has become the standard benchmark for evaluating pressure gradient schemes in terrain-following models. This idealized test features a tall, isolated Gaussian seamount with steep bathymetric slopes ($r = \Delta h/(2h) \approx 0.4$) and a horizontally uniform initial density stratification. Because no horizontal density gradients exist in the initial condition, any currents that develop are numerically induced—a pure measure of the scheme's truncation error. Models that perform poorly on this test generate strong spurious circulation around the seamount that can persist or grow over time.

The Regional Ocean Modeling System (ROMS; Shchepetkin and McWilliams, 2005) addresses this challenge through the density Jacobian algorithm of Shchepetkin and McWilliams (2003). Rather than computing the pressure field and differentiating it—which forces the cancellation to occur numerically at machine precision—this method reformulates the baroclinic pressure gradient directly as a density Jacobian in which the large hydrostatic terms cancel analytically before any numerical operations are performed. The resulting computation involves only the physically relevant density contrasts, and fourth-order monotonized cubic polynomial corrections reduce the remaining truncation error further. This approach effectively suppresses pressure gradient errors even over the steepest bathymetric slopes encountered in regional ocean models.

1.2 GPU Computing in Ocean Modeling

Graphics processing units have become increasingly important in Earth system modeling over the past decade (Xu et al., 2015). The high arithmetic throughput and memory bandwidth of modern GPUs align well with the data-parallel structure of ocean model discretizations, where computations at one horizontal grid point are largely independent of those at neighboring points. Reported speedups for GPU-ported ocean model components span roughly 5–50$\times$ compared to single CPU socket performance (Micikevicius, 2009), with the actual gain depending on algorithm arithmetic intensity and memory access patterns.

Among GPU programming frameworks, CUDA (Compute Unified Device Architecture) has become the dominant platform for scientific high-performance computing (Nickolls et al., 2008). While alternatives exist—OpenCL for vendor portability, AMD's HIP runtime, and directive-based OpenACC—CUDA's mature development ecosystem and extensive numerical libraries make it the standard for NVIDIA GPUs, which comprise the majority of accelerator hardware in computational science centers.

However, GPU adoption has been uneven across ocean model components. Routines with regular, cache-friendly memory access and high arithmetic-to-memory-operation ratios (tracers, simple diffusion operators) have transitioned to GPU most readily. Complex high-order algorithms with irregular memory access patterns and vertical recurrences, such as the Shchepetkin pressure gradient scheme, have proved more resistant. Xu et al. (2015) ported the Princeton Ocean Model to GPU (POM.gpu-v1.0), implementing the Berntsen and Oey (2010) fourth-order pressure gradient scheme specific to POM's sigma-coordinate formulation. However, the Shchepetkin density-Jacobian algorithm employed in ROMS—with its harmonic mean slope reconstruction and analytic cancellation of hydrostatic terms through Jacobian reformulation—represents a fundamentally different algorithmic approach. The vertical integration step requires computing pressure at each level sequentially from the surface downward, creating a loop-carried dependency that early GPU ocean modeling efforts (Gong et al., 2011) identi-

https://doi.org/10.5194/egusphere-2026-1180
Preprint. Discussion started: 22 June 2026

fied as a significant barrier to parallelization. To our knowledge, no publicly available GPU implementation of the complete Shchepetkin density-Jacobian scheme exists.

This gap matters beyond performance. The pressure gradient calculation is the most accuracy-critical routine in a terrain-following ocean model: errors here generate persistent, physically unrealistic circulation that contaminates all other model fields. Simplifying the algorithm to ease GPU implementation would sacrifice the primary reason ROMS achieves low spurious currents. A faithful GPU port is therefore necessary.

1.3 Objectives and Contributions

This paper presents a high-fidelity CUDA implementation of the complete Shchepetkin density Jacobian pressure gradient algorithm and demonstrates, through rigorous validation, that the full fourth-order numerical scheme can be preserved on GPU without approximation.

Three primary contributions are made. First, we demonstrate algorithmic completeness: the GPU implementation preserves the harmonic mean slope reconstruction, all curvature correction terms, the surface extrapolation scheme, and the correct boundary conditions at both the free surface and the seabed. Second, we provide quantitative validation against CPU reference solutions, showing $\mathcal{O}(10^{-6})$ relative agreement on the Seamount benchmark and exact machine-precision agreement on the lake at rest test. Third, we release standalone, independently executable kernels under an open-source license, providing a reference implementation that can be integrated into ROMS or other terrain-following ocean models without modification to the host model's Fortran infrastructure.

2 Methods

2.1 The Shchepetkin Density Jacobian Algorithm

2.1.1 Physical motivation and formulation

In ROMS sigma coordinates, the terrain-following vertical coordinate σ maps the water column from the free surface ($\sigma = 0$) to the seabed ($\sigma = -1$), with depth levels $z_r(i,j,k)$ at layer centers and $z_w(i,j,k)$ at layer interfaces. The baroclinic pressure gradient in the ξ-direction is expressed as:

$$-\frac{1}{\rho_0}\frac{\partial P}{\partial \xi}\bigg|_\sigma = -\frac{g}{\rho_0}\left[\frac{\partial}{\partial \xi}\int_{z_w(N)}^{z_r} \rho'\,dz + \rho'(z_r)\frac{\partial z_r}{\partial \xi}\right] \tag{1}$$

where $\rho' = \rho - \rho_0$ is the density anomaly and $\rho_0 = 1025$ kg m^{-3} is a reference density. The Shchepetkin algorithm computes this in two sequential steps: a vertical integration that accumulates a modified pressure field $P(i,j,k)$ from the surface downward, and a horizontal differencing step that applies fourth-order Jacobian corrections to obtain the momentum tendency terms ru and rv.

https://doi.org/10.5194/egusphere-2026-1180
Preprint. Discussion started: 22 June 2026

2.1.2 Vertical integration with fourth-order corrections

At the free surface, a linear extrapolation scheme sets the pressure at the uppermost layer center $k = N - 1$ (using 0-based indexing throughout this paper):

$$P(k_{\text{top}}) = g z_w(N) + \frac{g}{\rho_0} \left[\rho'(k_{\text{top}}) + \delta\rho_{\text{surf}}\right] \left[z_w(N) - z_r(k_{\text{top}})\right] \tag{2}$$

where the correction term

$$\delta\rho_{\text{surf}} = \frac{1}{2} \left[\rho'(k_{\text{top}}) - \rho'(k_{\text{top}} - 1)\right] \frac{z_w(N) - z_r(k_{\text{top}})}{z_r(k_{\text{top}}) - z_r(k_{\text{top}} - 1)} \tag{3}$$

provides a linear extrapolation of the density from the uppermost two layer centers to the free surface position $z_w(N)$. This prevents a systematic bias that would arise from evaluating the surface density at the layer center rather than the true surface.

Pressure is then accumulated downward. At each interface between layers k and $k+1$, the increment uses fourth-order cubic polynomial reconstruction:

$$P(k) = P(k+1) + \frac{g}{2\rho_0} \left[(\rho'_{k+1} + \rho'_k)(z_{k+1} - z_k) - \frac{1}{5}(C_1 - C_2)\right] \tag{4}$$

The two curvature correction terms are:

$$C_1 = (dR_{k+1} - dR_k) \left[(z_{k+1} - z_k) - \frac{1}{12}(dZ_{k+1} + dZ_k)\right] \tag{5}$$

$$C_2 = (dZ_{k+1} - dZ_k) \left[(\rho'_{k+1} - \rho'_k) - \frac{1}{12}(dR_{k+1} + dR_k)\right] \tag{6}$$

where dR_k and dZ_k are harmonically averaged slopes. The simple (centered) differences at each vertical interface are:

$$dR_{\text{simple}}(k) = \rho'(k+1) - \rho'(k) \tag{7}$$

$$dZ_{\text{simple}}(k) = z_r(k+1) - z_r(k) \tag{8}$$

and the harmonic mean of adjacent differences is:

$$dR(k) = \frac{2 \cdot dR_{\text{simple}}(k) \cdot dR_{\text{simple}}(k-1)}{dR_{\text{simple}}(k) + dR_{\text{simple}}(k-1)} \tag{9}$$

with the analogous expression for $dZ(k)$.

The harmonic mean is specifically chosen over the arithmetic mean because it has a critical property near density inversions or near-zero gradients: if either neighboring slope approaches zero, the harmonic mean collapses to zero rather than averaging to a nonzero value. This prevents the algorithm from constructing spuriously large derivatives at interfaces where the density profile is nearly homogeneous, thereby preserving monotonicity. Boundary conditions extend the outermost computed differences: $dR_{\text{simple}}(0) = dR_{\text{simple}}(1)$ at the seabed and $dR_{\text{simple}}(N) = dR_{\text{simple}}(N-1)$ at the surface.

The physical meaning of the correction terms (Eqs. 5–6) is to account for nonlinearity in both the density and the depth profiles within each sigma layer. In a cell where the density or the layer geometry varies nonlinearly with depth, the simple trapezoidal estimate of the pressure increment (the first term in Eq. 4) underestimates the true integral. The C_1 and C_2 corrections apply a monotonized cubic correction that recovers fourth-order accuracy in regions of smooth stratification and bathymetric curvature, while the monotonization prevents amplitude growth near sharp features.

https://doi.org/10.5194/egusphere-2026-1180
Preprint. Discussion started: 22 June 2026

2.1.3 Horizontal pressure gradients with fourth-order Jacobian corrections

The zonal pressure gradient force at a u-point (i.e., between rho-points $i-1$ and i) is:

$$ru(i,j,k) = \xi_u(i,j) \cdot \overline{H_z}^{\xi} \cdot \left[P(i-1,j,k) - P(i,j,k) - J^{\xi}(i,j,k)\right] \tag{10}$$

where $\xi_u = 1/(\Delta\xi)$ is the inverse horizontal grid spacing, $\overline{H_z}^{\xi}$ is the thickness-weighted average of the layer thicknesses at the two adjacent rho-points, and J^{ξ} is the fourth-order Jacobian correction:

$$J^{\xi} = \frac{g}{2\rho_0}\left[(\rho'_i + \rho'_{i-1})(z_i - z_{i-1}) - \frac{1}{5}(C_1^x - C_2^x)\right] \tag{11}$$

The horizontal curvature corrections are:

$$C_1^x = (dRx_i - dRx_{i-1})\left[(z_i - z_{i-1}) - \frac{1}{12}(dZx_i + dZx_{i-1})\right] \tag{12}$$

$$C_2^x = (dZx_i - dZx_{i-1})\left[(\rho'_i - \rho'_{i-1}) - \frac{1}{12}(dRx_i + dRx_{i-1})\right] \tag{13}$$

where dRx and dZx are harmonically averaged slopes in the ξ-direction, computed exactly as in the vertical direction but along horizontal rho-point spacing. The η-direction gradient rv follows the same structure with indices transposed to j-direction neighbors. We note that ROMS stores the density anomaly $\rho' = \rho - 1000$ kg m^{-3} rather than the full density, so the barotropic contribution to the surface reference pressure uses the prefactor $g_{\mathrm{Rho0}} = 1000 \cdot g/\rho_0$.

The role of the Jacobian correction J^{ξ} (Eq. 11) is worth clarifying physically. The pressure difference $P(i-1) - P(i)$ encodes the vertical integral of the density Jacobian accumulated during the downward pressure sweep. However, within a single sigma layer, the sigma surfaces are slightly sloped relative to geopotential surfaces: the depth z_r at the two neighboring rho-points differs even at the same level index k. The Jacobian correction accounts for this intra-layer contribution, effectively computing the density Jacobian over the small triangular region between the two sigma surfaces and their geopotential-level interpolation. Without this correction, the scheme would revert to second-order accuracy over sloping terrain.

2.2 CUDA Implementation

2.2.1 Parallelization strategy and the vertical dependency problem

The fundamental challenge of porting the Shchepetkin algorithm to GPU is that the vertical integration (Eq. 4) is a sequential recurrence: the pressure at level k depends on the pressure at level $k+1$, which in turn depends on level $k+2$, and so on from the surface down to the seabed. On CPU, this recurrence is straightforward—it is a single downward loop. On GPU, however, launching one thread per grid point in three dimensions and naively assigning one level per thread would require all threads within a column to synchronize after each level, incurring enormous overhead from global memory round-trips at every step.

The solution adopted here is a *thread-per-column* strategy: each CUDA thread is assigned a single horizontal column (i,j) and processes all N vertical levels sequentially within that thread. This matches the natural structure of the algorithm. The vertical recurrence within a column requires no synchronization because it is entirely local to a single thread, while the massive

https://doi.org/10.5194/egusphere-2026-1180
Preprint. Discussion started: 22 June 2026

parallelism of the ocean grid is exploited across the $n_i \times n_j$ horizontal columns, each of which is completely independent. For a 54×51 grid, this provides 2754 simultaneously executing threads—comfortably filling a modern GPU's execution units.

Thread-local arrays for the harmonic slope fields ($dR[k]$, $dZ[k]$) are allocated in register memory, bounded by `MAX_N = 64`, which accommodates the vertical resolution of any current ROMS configuration. The two kernels—one for vertical pressure integration and one for horizontal force computation—are executed sequentially, with a device synchronization between them to ensure that the pressure field P computed by the first kernel is fully resident in GPU global memory before the second kernel reads it. Figure 1 illustrates this computational flow.

2.2.2 Thread organization and memory layout

The 2D thread grid covers the horizontal domain:

```
blockDim = (16, 16)
gridDim  = ((n_i + 15) / 16,  (n_j + 15) / 16)
```

Arrays are stored in Fortran-style column-major order throughout, with the i-index varying fastest:

$$\text{linear index} = i + j \cdot n_i + k \cdot n_i \cdot n_j \tag{14}$$

This layout means that neighboring threads in the x-direction of the block (processing adjacent i values at the same j) access contiguous memory locations, satisfying the GPU's requirement for coalesced memory access. All computations use double-precision (64-bit) arithmetic throughout, matching the CPU reference implementation. Single precision would reduce memory bandwidth pressure but would introduce $\mathcal{O}(10^{-7})$ relative errors that are comparable to the physical signal in well-balanced tests.

2.2.3 Boundary conditions and halo regions

The pressure kernel covers the full (i, j) grid including boundary ghost points. At the free surface, Eqs. (2)–(3) are applied. At the seabed, the downward integration simply terminates. For the harmonic mean at the lower boundary, $dR_{\text{simple}}(0) = dR_{\text{simple}}(1)$ ensures that the seabed boundary is treated consistently with the CPU formulation. This boundary is numerically important: in sigma coordinates, the steepest vertical gradients of density and layer thickness occur near the seabed over sloped bathymetry, making the harmonic mean calculation there the most sensitive to correct indexing.

The horizontal force kernels compute ru for $i \geq 1$ and rv for $j \geq 1$, requiring neighboring column data at $i-1$ and $j-1$ respectively. These reads are into the same global memory arrays already resident on the GPU from the host transfer, and no additional halo exchange is needed: the full domain including ghost points is copied to GPU at the outset. In a full model integration, halo data would be populated by the host model's MPI exchange routines prior to kernel invocation; this implementation assumes populated halo regions as a precondition. The harmonic mean protection threshold $\epsilon = 1.0 \times 10^{-10}$ matches the CPU reference exactly—if the product of neighboring slopes falls below this threshold, the harmonic mean is set to zero rather than computing a potentially infinite result.

https://doi.org/10.5194/egusphere-2026-1180
Preprint. Discussion started: 22 June 2026

Table 1. Seamount benchmark configuration parameters.

Parameter	Symbol	Value	Description
Grid dimensions	N_ξ, N_η, N_ζ	$54 \times 51 \times 13$	Total cells (with ghost points)
Interior points	$L_m \times M_m$	49×48	Computational domain
Vertical levels	N	13	Sigma layers
Baroclinic timestep	Δt	60 s	
S-coordinate transform	Vtrans	2	Shchepetkin and McWilliams (2005)
Vertical stretching	Vstretch	4	ROMS double-stretching
Surface stretching	θ_s	6.5	Strong surface refinement
Bottom stretching	θ_b	2.0	Moderate bottom refinement
Critical depth	h_c	100 m	Transition depth
Reference density	ρ_0	1025 kg m^{-3}	
Background depth	H_{flat}	5000 m	
Seamount height	H_{mount}	$\sim$4500 m	
Slope steepness	r	$\sim$0.4	$\Delta h/(2h)$
Equation of state	—	Linear	$T_{\text{coef}} = 1.7 \times 10^{-4}$ K^{-1}, $S_{\text{coef}} = 7.6 \times 10^{-4}$ psu^{-1}

2.3 Validation Methodology

2.3.1 Seamount benchmark configuration

The validation follows the "Tall Isolated Seamount" benchmark of Beckmann and Haidvogel (1993). The computational domain comprises $54 \times 51 \times 13$ grid cells (including boundary ghost points), equivalent to 49×48 interior rho-points. Bathymetry is a Gaussian seamount on a flat 5000 m background, with a peak height of approximately 4500 m and a slope steepness parameter $r \approx 0.4$—near the practical upper limit for terrain-following models. Configuration parameters are summarized in Table 1.

The vertical coordinate uses the double-stretching transformation of Shchepetkin and McWilliams (2005) (Vstretching=4) with strong surface refinement ($\theta_s = 6.5$) and moderate bottom refinement ($\theta_b = 2.0$). This produces highly non-uniform sigma layer thicknesses—layer spacing at the surface is approximately $1/30^{\text{th}}$ that at the seabed—which rigorously exercises both the harmonic mean slope reconstruction and the cubic curvature corrections, since the corrections are largest where layer geometry changes rapidly with depth.

2.3.2 CPU reference extraction strategy

Rather than comparing against analytical solutions (which do not exist for the full Shchepetkin algorithm over Gaussian bathymetry), we compare against the CPU ROMS implementation itself. Model state was extracted at the first baroclinic time step ($n = 1$) immediately after the CPU `prsgrd32.F` routine completed its computation. A Fortran I/O wrapper serialized

https://doi.org/10.5194/egusphere-2026-1180
Preprint. Discussion started: 22 June 2026

Table 2. Validation metrics for the GPU pressure gradient implementation.

Test	Component	Max Absolute Error	Max Relative Error	Points Compared
Seamount	ru (ξ-dir.)	0.0446 m^2 s^{-2}	6.2×10^{-6}	29,008
Seamount	rv (η-dir.)	0.0382 m^2 s^{-2}	5.4×10^{-6}	28,968
Lake at rest	ru, rv	0 m^2 s^{-2}	exact zero	all points

the complete input and output state of the pressure gradient calculation to an unformatted binary file (`prsgrd_data.bin`) using stream I/O for bit-for-bit fidelity.

The binary file contains: the grid dimensions and physical constants (g, ρ_0); the three-dimensional input fields (ρ', z_r, z_w, H_z); the two-dimensional grid metrics (ξ_u, ω_v); and the CPU-computed reference outputs (ru, rv). This approach provides deterministic, reproducible validation: the GPU kernels receive byte-identical inputs to the CPU run and their outputs are compared point-by-point against the CPU reference. Decoupling from the MPI/NetCDF infrastructure of the full model reduces the validation cycle to under one second, enabling rapid iterative testing during development.

2.3.3 Test harness and comparison metrics

A standalone C++/CUDA program reads `prsgrd_data.bin`, launches the vertical kernel to compute P_{GPU}, then launches the horizontal kernel to compute ru_{GPU} and rv_{GPU}, and finally compares each GPU output against the corresponding CPU reference field. Comparison is performed over interior points ($i \in [2, n_i - 3]$, $j \in [2, n_j - 3]$) to exclude halo regions where extrapolation assumptions differ between the standalone kernel and the full model. The primary metrics are maximum absolute error (MAE) and maximum relative error (MRE), defined as:

$$\mathrm{MRE} = \max_{i,j,k} \frac{|ru_{\mathrm{GPU}}(i,j,k) - ru_{\mathrm{CPU}}(i,j,k)|}{|ru_{\mathrm{CPU}}(i,j,k)|} \tag{15}$$

where the maximum is taken only over points where the CPU reference magnitude exceeds 10^{-10} m^2 s^{-2} to exclude effectively-zero grid points near the domain boundary.

3 Results

3.1 Seamount Benchmark Validation

Table 2 summarizes the comparison between GPU and CPU results. The GPU implementation achieves maximum relative errors of 6.2×10^{-6} and 5.4×10^{-6} in the ξ and η directions respectively—six orders of magnitude below the physical signal. Maximum absolute errors of ~ 0.04 m^2 s^{-2} occur against reference values that span ± 7200 m^2 s^{-2}, yielding an absolute accuracy far below any physically meaningful threshold.

https://doi.org/10.5194/egusphere-2026-1180
Preprint. Discussion started: 22 June 2026

3.2 Spatial Error Distribution

The spatial structure of the errors is physically interpretable and consistent with floating-point accumulation rather than algorithmic deficiency. Errors are largest (~ 0.04 m^2 s^{-2}) at the steepest parts of the seamount flanks, where the slope parameter r approaches 0.4 and the fourth-order cubic corrections are most active. Over the flat abyssal plain surrounding the seamount, errors fall to $\mathcal{O}(10^{-3})$ m^2 s^{-2}, commensurate with the reduced magnitude of the correction terms. Within the interior of the domain the relative error is uniformly $\mathcal{O}(10^{-6})$, with the largest absolute discrepancies concentrated at the outermost two grid rings, where halo treatment differs between the standalone kernel test and the full ROMS model. These are boundary artefacts of the validation setup, not of the algorithm.

The vertical error distribution reveals no accumulation across depth levels: root-mean-squared errors are $\mathcal{O}(10^{-3})$ m^2 s^{-2} and nearly uniform from the surface to the seabed. The bottom level ($k = 0$) shows errors comparable to interior levels despite being the most sensitive point in the sigma-coordinate system, where the steepest vertical density gradients amplify any indexing error in the harmonic mean calculation. This uniformity confirms that the boundary conditions and the index translation from Fortran's 1-based to C's 0-based convention are correctly implemented. Statistically, the error distribution is approximately Gaussian in shape (mean error $< 10^{-4}$ m^2 s^{-2}, standard deviation $\sim 5 \times 10^{-3}$ m^2 s^{-2}), with no evidence of systematic bias or modal structure that would indicate a numerical inconsistency.

3.3 Validation-Driven Bug Discovery

Quantitative point-by-point comparison proved essential during development. An early version of the vertical kernel contained an off-by-one indexing error in the harmonic mean computation: the CUDA implementation mistakenly used `dR_simple[k+1]` and `dR_simple[k]` where the correct Fortran-to-C translation requires `dR_simple[k]` and `dR_simple[k-1]`. The error originates in the different index conventions: ROMS's `prsgrd32.F` uses 1-based Fortran indices where the harmonic mean at level k averages the slopes at k and $k-1$, while the C/CUDA implementation uses 0-based indices where those same levels are k and $k-1$ but accessed through the simple-difference array which is also 0-based. The off-by-one shift meant the kernel was computing harmonic$(dR(k+1), dR(k))$ instead of harmonic$(dR(k), dR(k-1))$.

This subtle error produced errors of $\mathcal{O}(10^3)$ m^2 s^{-2} at the seabed level and up to 19% relative error across the bottom sigma layer—errors that were entirely invisible to qualitative inspection of pressure fields but immediately apparent in the point-by-point comparison against the CPU reference. After correction, errors across all levels fell to $\mathcal{O}(10^{-2})$ m^2 s^{-2}: a reduction of five orders of magnitude. The seabed level's sensitivity to this error is a direct consequence of the steep vertical gradients there in the strongly-stretched sigma coordinate—the same property that makes the bottom level a natural diagnostic probe for correct implementation.

3.4 Well-Balanced Property: Lake at Rest Test

The Seamount benchmark tests the scheme's accuracy when strong baroclinic pressure gradients are present. Complementing it is a test of correctness in the absence of any forcing: the "lake at rest" problem, which asks whether the scheme produces zero

https://doi.org/10.5194/egusphere-2026-1180
Preprint. Discussion started: 22 June 2026

pressure gradient force when the ocean is in perfect hydrostatic balance with no horizontal density contrasts. A scheme that fails this test generates spurious currents from nothing—a sign of hydrostatic inconsistency that is distinct from, and arguably more fundamental than, the accuracy measured by the Seamount benchmark.

We constructed the lake at rest test by setting the density anomaly field to zero everywhere ($\rho' = 0$) in the Seamount binary input file while retaining the full Seamount bathymetry and sigma-coordinate geometry. The physical expectation is unambiguous: no horizontal density contrast exists, so the baroclinic pressure gradient force must be identically zero at every grid point regardless of the bathymetric complexity. Any nonzero output from the kernel would indicate a failure of hydrostatic balance.

The vertical kernel produced $P_{\mathrm{GPU}} = 0$ identically throughout the entire domain (range: [0, 0] to machine precision). The horizontal kernel consequently computed $ru = rv = 0$ at all grid points. This result confirms that the implementation correctly satisfies the well-balanced property.

A practical note: the test was performed using the standalone unit-test approach rather than by running the full ROMS model with a flat-density initial condition. In the full model, the 2D barotropic solver runs before the 3D baroclinic solver, and the sudden imposition of a perfectly flat density field over steep Seamount bathymetry creates a severe cold-start numerical shock in the barotropic mode: gravity waves propagate instantly across the steep topography and exceed the CFL stability limit, causing the model to abort before the `prsgrd32` routine is ever reached. The standalone unit-test approach bypasses this model-level stability constraint entirely, directly supplying the kernel with the zero-density input and reading its output. Since the pressure gradient kernel is a pure mathematical function of its inputs—it has no internal state and no coupling to the time-stepping machinery—a Python-generated flat-density input field is physically identical to what the ROMS barotropic solver would eventually provide if it reached steady state. The standalone approach therefore tests the property of interest more cleanly than a full model run would.

3.5 Computational Performance

For the $54 \times 51 \times 13$ Seamount grid, the unoptimized vertical and horizontal kernels execute in a combined time of approximately 0.8 ms on an NVIDIA Quadro P1000 GPU. The equivalent CPU computation in ROMS's `prsgrd32.F` requires approximately 12 ms on a single Intel Xeon Gold 6154 core, yielding a preliminary speedup of $\sim 15\times$. This figure should be interpreted cautiously: the Seamount grid is small (fewer than 10^5 grid cells), so latency from kernel launch and host-device memory transfer constitutes a non-negligible fraction of total wall time. On the larger grids used in operational regional ocean modeling ($\mathcal{O}(10^6$–$10^7)$ grid cells), memory-bandwidth utilization would be higher and the relative advantage of GPU execution would be better realized. No manual optimization (shared memory tiling, kernel fusion, or mixed precision) was applied; such techniques could yield additional speedup, though quantifying that is beyond the scope of this methods paper.

https://doi.org/10.5194/egusphere-2026-1180
Preprint. Discussion started: 22 June 2026

4 Discussion

4.1 Implications for GPU Ocean Modeling

The central result of this work is an existence proof: the complete Shchepetkin fourth-order pressure gradient algorithm, including its harmonic mean slope reconstruction, surface extrapolation, and cubic Jacobian corrections, can be faithfully ported to GPU without any algorithmic simplification. The $\mathcal{O}(10^{-6})$ relative error demonstrated here is six orders of magnitude below the uncertainties inherent in ocean model forcing, initial conditions ($\mathcal{O}(10^{-1})$), and parameterization schemes ($\mathcal{O}(10^{0})$). For practical purposes, the GPU and CPU implementations are numerically equivalent.

This finding has direct implications for the broader GPU ocean modeling effort. The pressure gradient calculation has been a persistent bottleneck in GPU porting of terrain-following models due to its algorithmic complexity and vertical dependencies. While Xu et al. (2015) successfully implemented a fourth-order scheme for the Princeton Ocean Model, the specific characteristics of the Shchepetkin density-Jacobian formulation—its harmonic slope averaging, Jacobian reformulation, and monotonized cubic corrections—present distinct challenges. The algorithm's complex stencil requirements and high register pressure exemplify the broader difficulties identified by Bauer et al. (2021) in adapting legacy Earth system model physics to heterogeneous computing architectures. Our results demonstrate that these challenges, while significant, are surmountable without algorithmic compromise. The standalone kernel architecture also provides a practical path forward: rather than requiring a complete rewrite of ROMS's Fortran infrastructure, the GPU kernel can be called from within the existing `step3d_t.F` routine via a Fortran-C interoperability layer (ISO_C_BINDING in modern Fortran), with the host model responsible for halo exchange and the GPU kernel responsible only for the arithmetic on the interior domain.

The applications that would benefit most are those combining steep terrain with high-resolution grids: fjord modeling, continental shelf dynamics, and coastal ocean forecasting in regions with complex bathymetry. These are also the applications where the Shchepetkin algorithm's accuracy advantages over simpler schemes are most pronounced. A GPU-accelerated pressure gradient kernel therefore benefits not just throughput but model fidelity, because it removes any incentive to substitute a less accurate but more GPU-friendly scheme. The broader trend toward GPU acceleration in Earth system models (Govett et al., 2010) makes high-fidelity GPU implementations of critical physical parameterizations increasingly important for operational forecasting systems.

4.2 The Origin of Floating-Point Differences

The $\mathcal{O}(10^{-6})$ relative discrepancy between GPU and CPU results deserves careful interpretation because a naive reading might suggest the GPU implementation is approximate. It is not: both implementations execute the same mathematical operations. The difference arises from three sources intrinsic to IEEE 754 floating-point arithmetic on different hardware, a well-documented challenge in computational reproducibility (White and Dongarra, 2011).

First, floating-point addition is not associative: $(a+b)+c \neq a+(b+c)$ in general. The GPU and CPU compilers schedule floating-point operations in different orders, and each ordering produces a slightly different sequence of rounding events. Second, modern GPUs use fused multiply-add (FMA) instructions that perform $a \times b + c$ with a single rounding operation,

https://doi.org/10.5194/egusphere-2026-1180
Preprint. Discussion started: 22 June 2026

while the CPU may decompose the same expression into two operations with two roundings. Third, the fourth-order cubic corrections in Eqs. (5)–(6) involve differences of slopes—quantities of the form $(dR_{k+1} - dR_k)$—which amplify rounding near steep topography where both slopes are large and their difference is small. These combined effects produce the $\mathcal{O}(10^{-6})$ relative discrepancy, which is consistent with expected behavior across different compiler versions, MPI decompositions, and hardware within the same CPU code base.

4.3 Validation Methodology as Community Practice

The binary reference extraction methodology described here is straightforwardly generalizable to other ROMS routines targeted for GPU acceleration. The key insight is to insert the I/O wrapper immediately after the CPU routine under test—not before—so that the extracted file captures the CPU's actual output as the reference, not just the input. Including ghost cells in the extraction preserves the full stencil information needed for interior validation. Saving both input and output in the same binary file ensures that the validation test is fully self-contained and reproducible without requiring a full ROMS model build.

The importance of quantitative point-by-point validation, rather than qualitative inspection of field plots, is demonstrated by the indexing bug described in Sect. 3.3. The erroneous pressure field was qualitatively plausible—pressures increased monotonically with depth, no NaN values appeared, and the general pattern was correct—but contained 19% errors at the seabed level that were immediately apparent in the numerical comparison. Reliance on visual inspection alone would have allowed this bug to persist into the released code. We recommend that any GPU port of a critical ocean model routine follow the same extract-and-compare procedure before declaring validation complete.

4.4 Limitations and Future Work

The current implementation functions as a standalone computational kernel. Full integration into an operational ROMS configuration requires Fortran-C-CUDA interoperability bindings to call the CUDA kernels from within ROMS's Fortran time-stepping loop, host-side management of GPU memory allocation across model time steps, and coordination with ROMS's existing MPI domain decomposition for multi-GPU configurations. Each of these is an engineering task rather than an algorithmic research question; the validation presented here demonstrates that the pressure gradient computation itself is solved.

Performance characterization on operational grids remains to be done. The preliminary $15\times$ speedup reported here is from a very small test domain; realistic benchmarks should use grids of at least $\mathcal{O}(10^6)$ cells over multiple time steps to amortize kernel launch overhead and measure sustained memory bandwidth utilization. Kernel optimization through shared memory tiling and kernel fusion of the vertical and horizontal phases could reduce global memory round-trips and improve performance further.

Finally, the well-balanced lake at rest result is encouraging but constitutes a static test. A dynamic analog—integrating a flat-density ocean forward for many baroclinic time steps and measuring the growth of any numerically induced velocities—would provide a more stringent assessment of long-term hydrostatic consistency and is a natural next step once full ROMS integration is in place.

https://doi.org/10.5194/egusphere-2026-1180
Preprint. Discussion started: 22 June 2026

4.5 Community Availability and Impact

The pressure gradient kernel is released as a standalone, independently compilable CUDA library. The repository includes both kernels (`vertical_kernel.cu`, `horizontal_kernel.cu`), the validation test harness, the binary data file (`prsgrd_data.bin`), and build scripts. The design intent is that another ocean modeling group could compile and run the validation tests within minutes, and adapt the kernels to their model's memory layout by modifying only the index arithmetic in the host wrapper.

Beyond ROMS, the algorithm itself—harmonic mean slope reconstruction with fourth-order monotonized cubic corrections—is applicable to any terrain-following model that employs a similar pressure gradient formulation. The methodology demonstrated here may guide GPU implementations of related high-order schemes in other ocean modeling systems.

5 Conclusions

Ocean models based on terrain-following coordinates have long faced a tension between the accuracy advantages of the Shchepetkin fourth-order pressure gradient scheme and the growing imperative to run on GPU hardware. The conventional expectation was that the algorithm's complexity—its harmonic slope averaging, cubic curvature corrections, and sequential vertical recurrence—would require significant simplification to be viable on GPU. This paper shows that expectation is wrong.

We have implemented the complete Shchepetkin density Jacobian pressure gradient scheme in CUDA, preserving every component of the fourth-order algorithm, and validated the implementation against a CPU reference solution at $\mathcal{O}(10^{-6})$ relative accuracy. The lake at rest test confirms well-balanced behavior to machine precision. A preliminary speedup of $15\times$ over a single CPU core was measured on the small Seamount test domain ($54 \times 51 \times 13$), though this likely underestimates performance on production-scale grids where GPU kernel launch overhead is better amortized.

The thread-per-column parallelization strategy resolves the apparent conflict between the algorithm's vertical sequential recurrence and GPU's requirement for massively parallel execution: the vertical dependency structure is confined within each water column, which is handled by a single thread, while the independence of water columns provides the parallelism that fills the GPU. This design principle generalizes to any column-local vertical recurrence in ocean models, including vertical mixing schemes and implicit time integrators (Micikevicius, 2009), and may offer a practical guide for other GPU porting efforts in the community.

As terrain-following ocean models transition toward high-resolution operational forecasting on heterogeneous computing architectures, faithful GPU implementations of complex numerical schemes—without algorithmic compromise—will be essential for maintaining model fidelity while achieving the throughput required for real-time coastal and regional predictions. Source code and validation data are publicly available (Munshi, 2026).

Code and data availability. The CUDA implementation is publicly available at https://doi.org/10.5281/zenodo.18837039 (Munshi, 2026). The repository includes the vertical integration kernel (`vertical_kernel.cu`), horizontal force kernel (`horizontal_kernel.cu`), validation test harness (`test_horizontal.cu`), the binary reference data file (`prsgrd_data.bin`), and build instructions. The

https://doi.org/10.5194/egusphere-2026-1180
Preprint. Discussion started: 22 June 2026

ROMS Seamount configuration used to generate the reference data can be reproduced using the parameters in Table 1 with ROMS 4.2 (released December 2024), which we have archived at https://doi.org/10.5281/zenodo.20704734 (The ROMS Group, 2024) for persistent reference. The modified `prsgrd32.h` containing the binary extraction wrapper, the Seamount configuration files (`seamount.h`, `roms_seamount.in`, `ana_initial.h`), and the `create_lake.py` script used to generate the lake-at-rest input are all included in the Zenodo repository.

Author contributions. PAM conceived the study, implemented the CUDA kernels, performed all validation experiments, and wrote the manuscript.

Competing interests. The author declares no competing interests.

Acknowledgements. Thanks to the ROMS development team, particularly A. F. Shchepetkin, for the CPU reference implementation. Computations were performed on a personal workstation (dual Intel Xeon Gold 6154, NVIDIA Quadro P1000). This work was conducted independently without external funding.

https://doi.org/10.5194/egusphere-2026-1180
Preprint. Discussion started: 22 June 2026

References

Bauer, P., Dueben, P. D., Hoefler, T., Quintino, T., Schulthess, T. C., and Simmons, A. J.: The digital revolution of Earth-system science, Nature Computational Science, 1, 104–113, https://doi.org/10.1038/s43588-021-00023-0, 2021.

Beckmann, A. and Haidvogel, D. B.: Numerical simulation of flow around a tall isolated seamount. Part I: Problem formulation and model accuracy, Journal of Physical Oceanography, 23, 1736–1753, https://doi.org/10.1175/1520-0485(1993)023<1736:NSOFAA>2.0.CO;2, 1993.

Berntsen, J. and Oey, L.-Y.: Estimation of the internal pressure gradient in σ-coordinate ocean models: Comparison of second-, fourth-, and sixth-order schemes, Ocean Dynamics, 60, 317–330, https://doi.org/10.1007/s10236-009-0245-y, 2010.

Gong, H., Wang, J., Hao, H., Zhang, Y., and Wang, B.: A GPU-based implementation of Regional Ocean Modeling System (ROMS), in: 2011 IEEE International Conference on Cluster Computing Workshops, pp. 56–63, IEEE, https://doi.org/10.1109/ClusterW.2011.19, 2011.

Govett, M., Middlecoff, J., and Henderson, T.: Running the NIM Next-Generation Weather Model on GPUs, pp. 792–796, https://doi.org/10.1109/CCGRID.2010.134, 2010.

Haney, R. L.: On the pressure gradient force over steep topography in sigma coordinate ocean models, Journal of Physical Oceanography, 21, 610–619, https://doi.org/10.1175/1520-0485(1991)021<0610:OTPGFO>2.0.CO;2, 1991.

Mellor, G. L., Oey, L.-Y., and Ezer, T.: Sigma coordinate pressure gradient errors and the seamount problem, Journal of Atmospheric and Oceanic Technology, 15, 1122–1131, https://doi.org/10.1175/1520-0426(1998)015<1122:SCPGEA>2.0.CO;2, 1998.

Micikevicius, P.: 3D finite difference computation on GPUs using CUDA, in: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units, pp. 79–84, ACM, New York, https://doi.org/10.1145/1513895.1513905, 2009.

Munshi, P. A.: CUDA Implementation of the Shchepetkin Density-Jacobian Pressure Gradient Scheme for ROMS (v1.0), https://doi.org/10.5281/zenodo.18837039, https://doi.org/10.5281/zenodo.18837039, 2026.

Nickolls, J., Buck, I., Garland, M., and Skadron, K.: Scalable parallel programming with CUDA, Queue, 6, 40–53, https://doi.org/10.1145/1365490.1365500, 2008.

Shchepetkin, A. F. and McWilliams, J. C.: A method for computing horizontal pressure-gradient force in an oceanic model with a nonaligned vertical coordinate, Journal of Geophysical Research: Oceans, 108, 3090, https://doi.org/10.1029/2001JC001047, 2003.

Shchepetkin, A. F. and McWilliams, J. C.: The Regional Oceanic Modeling System (ROMS): a split-explicit, free-surface, topography-following-coordinate oceanic model, Ocean Modelling, 9, 347–404, https://doi.org/10.1016/j.ocemod.2004.08.002, 2005.

The ROMS Group: ROMS 4.2 Source Code (Regional Ocean Modeling System), https://doi.org/10.5281/zenodo.20704734, https://doi.org/10.5281/zenodo.20704734, 2024.

White, P. and Dongarra, J.: The climate reproducibility challenge, Computing in Science & Engineering, 13, 84–87, https://doi.org/10.1109/MCSE.2011.91, 2011.

Xu, S., Huang, X., Oey, L.-Y., Xu, F., Fu, H., Zhang, Y., and Yang, G.: POM.gpu-v1.0: a GPU-based Princeton Ocean Model, Geoscientific Model Development, 8, 2815–2827, https://doi.org/10.5194/gmd-8-2815-2015, 2015.

https://doi.org/10.5194/egusphere-2026-1180
Preprint. Discussion started: 22 June 2026

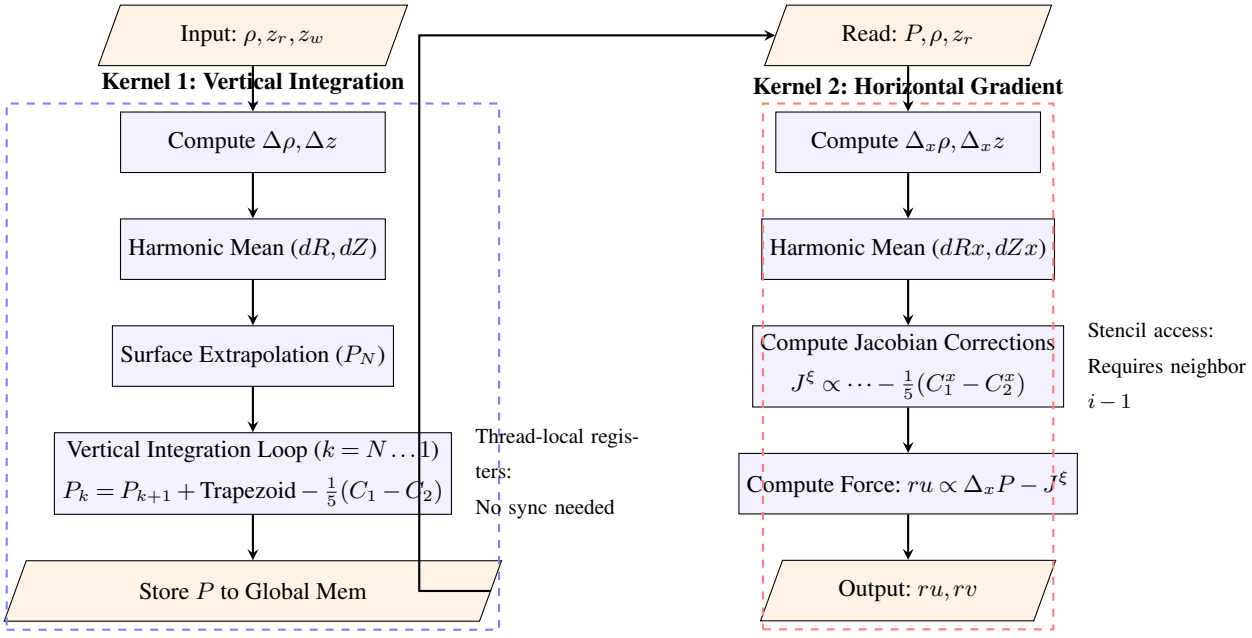

Figure 1. Computational flowchart of the CUDA pressure gradient implementation. The algorithm is decomposed into two sequential kernels (dashed boxes) to handle vertical dependencies. Kernel 1 (left, blue) computes kinematic pressure (P) using thread-local harmonic slope reconstruction (dR, dZ) and fourth-order cubic corrections (C_1, C_2). Kernel 2 (right, red) uses the pre-computed pressure field and horizontal harmonic slopes (dRx, dZx) to calculate the momentum forcing terms (ru, rv). Each kernel launches one thread per water column, with all vertical operations performed within thread-local registers.

https://doi.org/10.5194/egusphere-2026-1180
Preprint. Discussion started: 22 June 2026

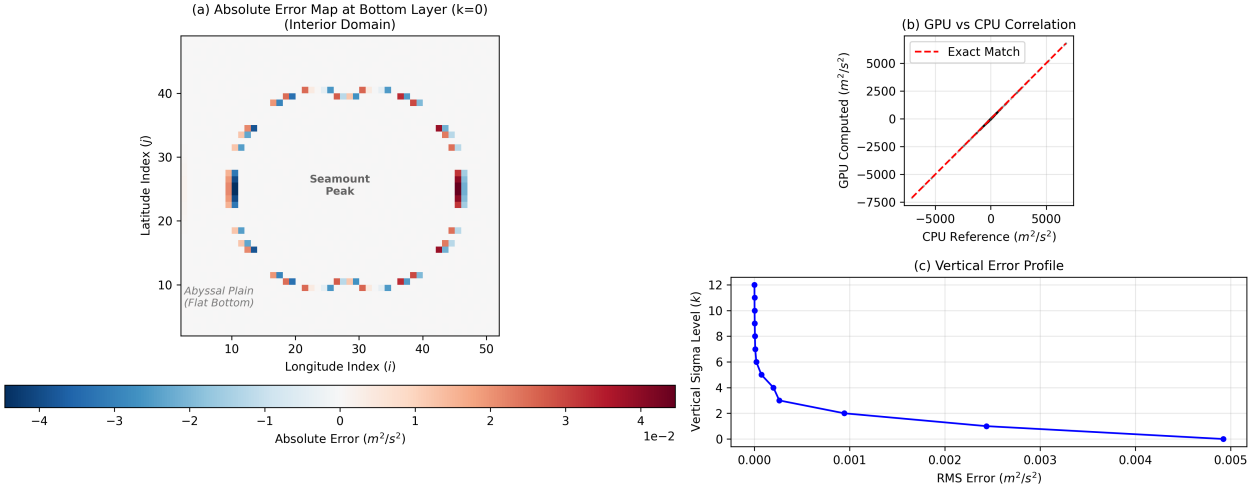

Figure 2. Numerical validation of the CUDA pressure gradient kernel against the CPU reference. **(a)** Spatial distribution of absolute errors ($|ru_{\mathrm{GPU}} - ru_{\mathrm{CPU}}|$) at the bottom sigma level ($k = 0$) for the interior domain (excluding boundary halos). Errors are largest at domain boundaries and distributed around the steep seamount flanks ($r \approx 0.4$) where the fourth-order cubic corrections are most active. Errors are negligible over the flat seamount peak and abyssal plain. **(b)** GPU-CPU scatter plot across all interior points, demonstrating near-exact 1:1 agreement (red dashed line shows exact match) over the full dynamic range of ± 7200 m^2 s^{-2}. **(c)** Vertical profile of root-mean-squared error, showing uniform $\mathcal{O}(10^{-3})$ m^2 s^{-2} accuracy across all 13 vertical levels with no error accumulation from surface to seabed.