A GPU-based ocean dynamical core for routine mesoscale-resolving climate simulations

Simone Silvestri[1], Gregory L. Wagner[1], Navid C. Constantinou[2,3], Christopher N. Hill[1], Jean-Michel Campin[1], Andre N. Souza[1], Siddhartha Bishnu[1], Valentin Churavy[1], John Marshall[1], and Raffaele Ferrari[1]

[1]Massachusetts Institute of Technology, Cambridge, MA, USA
[2]University of Melbourne, Parkville, VIC, Australia
[3]ARC Center of Excellence for the Weather of the 21st Century, Parkville, VIC, Australia

Key Points:

- A novel, GPU-tailored algorithm for finite-volume ocean dynamical cores yields unprecedented time-to-solution
- By following GPU-specific implementation recipes, it is possible to obtain efficient dynamical cores for GPU.
- Routine mesoscale-resolving climate simulations are feasible with GPU-based ocean models

Abstract

We describe an ocean hydrostatic dynamical core implemented in Oceananigans optimized for Graphical Processing Unit (GPU) architectures. On 64 A100 GPUs, equivalent to 16 computational nodes in current state-of-the-art supercomputers, our dynamical core can simulate a decade of near-global ocean dynamics per wall-clock day at an 8-kilometer horizontal resolution; a resolution adequate to resolve the ocean's mesoscale eddy field. Such efficiency, achieved with relatively modest hardware resources, suggests that climate simulations on GPUs can incorporate fully eddy-resolving ocean models. This removes a major source of systematic bias in current IPCC coupled model projections, the parameterization of ocean eddies, and represents a major advance in climate modeling. We discuss the computational strategies, focusing on GPU-specific optimization and numerical implementation details that enable such high performance.

Plain Language Summary

State-of-the-art ocean models used in climate studies cannot resolve small-scale turbulent features like eddies, which are important for accurate climate projections. We introduce a new ocean dynamical core implemented in the Julia library Oceananigans, designed to run efficiently on Graphical Processing Units (GPUs). Using relatively modest hardware resources, this model can simulate a decade of global ocean dynamics in a day at a scale that resolves turbulent eddies. This efficiency suggests that climate simulations on GPUs could transition to fully resolving ocean eddies, which are currently only partially captured due to computational limitations on Central Processing Units (CPUs). Resolving these eddies is expected to improve the accuracy of climate projections by addressing biases associated with the poor representation of ocean eddies. We discuss the computational strategies and implementation details behind this high performance.

Corresponding author: Simone Silvestri, `silvestri.simone0@gmail.com`

1 Introduction

Most climate projections use ocean components with a lateral resolution of 25 to 100 kilometers. With such coarse resolutions, the most energetic features of Earth's ocean — such as the Gulf Stream, or the Southern Ocean's rich field of mesoscale eddies — are either completely unresolved or, at best, partially resolved (Hewitt et al., 2020). As a result, these crucial features must be fully or partly represented by approximate parameterizations (Gent & Mcwilliams, 1990), which compromise the fidelity of the simulated ocean circulation, the ocean uptake of atmospheric heat and carbon, and the overall accuracy of climate projections (e.g., Griffies et al. (2015); Roberts et al. (2018); Chassignet et al. (2020); Constantinou and Hogg (2021); Couespel et al. (2024)). In this paper, we describe a new ocean dynamical core, or "dycore" that is optimized for general-purpose Graphics Processing Units (GPUs). Leveraging GPUs allow us to run the dynamical core on modest compute resources with unprecedented time-to-solution, significantly improving the efficiency of ocean simulations. This step-change in efficiency means that higher-resolution ocean simulations for the same computational cost are possible — enabling climate projections that resolve, rather than parameterize, ocean mesoscale turbulence.

Eddy-resolving simulations of the global ocean, which require lateral resolutions of $O(10\,\mathrm{km})$ or finer (Hallberg, 2013), are now routine for scientific purposes (e.g. Kiss et al., 2020; Ding et al., 2022). But climate projections require ensembles of hundreds of simulations to calibrate climate model free parameters (Schneider et al., 2017), to explore outcomes under the range of plausible future emission scenarios, and to disentangle internal and forced variability (Kay et al., 2015). Using $O(10\,\mathrm{km})$ lateral resolution — 2–10$\times$ finer than the current state-of-the-art — increases computational costs by $\sim$10–100$\times$ due to the corresponding increase in both horizontal degrees of freedom and the smaller time-steps needed to simulate mesoscale turbulence. Finally, we note that while 2–10$\times$ increase in ocean model resolution yields major improvements by resolving a new regime of oceanic motion, the same is not true for a similar increase in atmospheric model resolution. For example, Palmer (2014) argues that atmospheric models require 1 km resolution to achieve a step change accuracy by resolving deep convection, 100$\times$ finer than the typical 100 km resolution used for climate projection. Since using 1 km atmospheric model resolution would require increasing computational efficiency 100$\times$ over the current state-of-the-art, major improvements to climate model fidelity cannot be achieved merely by optimizing an atmospheric dynamical core for GPUs.

CPU-based climate models have historically realized efficiency gains because of advances in CPU fabrication technology (Schaller, 1997). But because advances in fabrication technology have stagnated, the days of "free lunch" are over (Sutter et al., 2005). Fortunately, because CPUs are not purpose-designed for scientific computing — they are limited at a structural level by design choices that are specifically detrimental to structured computations like machine learning and climate modeling (Vance, 2009) — efficiency gains are achievable through other advances in processor design and instruction set architecture. Enter general-purpose GPUs, which represent a decade of such innovations targeting precisely the kinds of structured computations encountered in both machine learning *and* computational fluid dynamics. GPU-based advances in scientific computing both enabled (Raina et al., 2009; Krizhevsky et al., 2017) and continue to be driven by the ongoing AI revolution.

While there has been progress in developing GPU-based atmospheric dycores (Fuhrer et al., 2018; M. Taylor et al., 2023), the potential for GPUs to accelerate ocean dycores has received limited attention. In particular, most novel GPU atmospheric dycores solve the compressible form of the Navier Stokes equations, which benefits particularly from a spectral element discretization (Souza et al., 2023; Fuhrer et al., 2018; M. Taylor et al., 2023). The requirements for efficient GPU implementation are different for the compressible Navier-Stokes equations compared to the Primitive equations, typically solved in ocean dycores (for example, handling sound waves as opposed to having a free surface solver). One notable exception is the work by Kochkov et al. (2024), which presents a fully differentiable

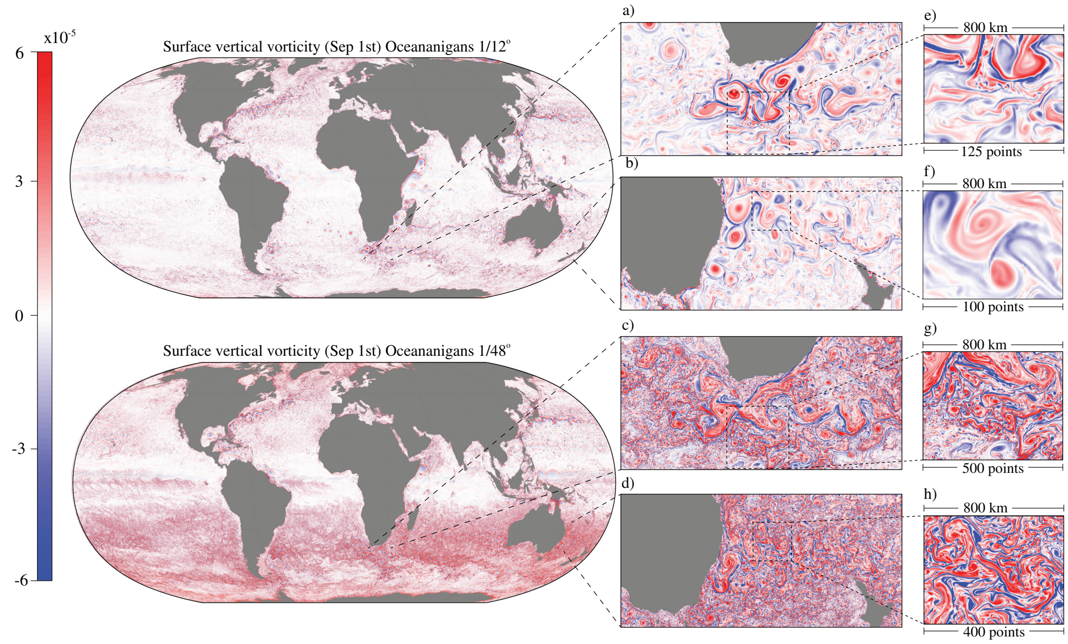

Figure 1: Vertical vorticity on on September 1st as simulated with the near-global configuration at a lateral resolution of $1/12°$ degree after 20 years of integration (top left) and at a $1/48°$ degree-resolution after a 1 year integration (bottom left). To the right, the insets zoom on particularly energetic regions of the ocean: the Aghulas and the East Australian Currents. While major ocean currents with widths of 10-100 km are resolved in both simulations, the sharp density fronts and associated currents that develop at the ocean surface in winter at scales between 1-10 km (the ocean weather) are only resolved by the model at a $1/48°$ lateral resolution. On September 1 — spring in the southern hemisphere and fall in the northern hemisphere — such sharp frontal features populate the Southern ocean, but are suppressed in the Northern hemisphere.

primitive equation atmospheric model written in JAX for TPUs. However, despite the use of primitive equations, Kochkov et al. (2024) uses spectral numerics that cannot be used in ocean models due to the presence of lateral boundaries. Regarding ocean dycores, P. Wang et al. (2021) document a translation of the LiCOM3 ocean model to GPUs, obtaining a speedup of 4× to 6× on a node with 4 GPUs compared to the CPU counterpart running on the same node with 32 CPU cores. However, given the difference in hardware and execution models between GPU and CPU, to achieve optimal performance both the model structure and the algorithmic implementation must be redesigned to adapt the model to the new architecture. Häfner et al. (2021) go one step further and design an ocean dycore called Veros for GPUs from scratch and achieve good computational performance. However, Veros was designed to be differentiable through the JAX framework, preventing granular performance optimization (Rackauckas, 2023).

In this paper, we take a different approach and implement, from a clean state, an algorithm for solving the hydrostatic Boussinesq equations in ocean dycores on GPUs with the objective of optimizing computational efficiency. The equations we implement, standard for ocean modeling, are written down in section 2. In section 3, we describe the implementation of the dycore, which includes numerical optimization and software design central to achieving performance on both single and multiple GPUs. In section 4, we describe a quasi-realistic near-global ocean setup that we use to test the algorithm's performance. The performance results, described in section 5, are promising: at a horizontal resolution of 1/12th degree our dycore achieves 10 simulated years per day (SYPD) on just 64 Nvidia A100 GPUs. A visualization of the solution is shown in figure 1. Section 6 showcases solutions of the particular mesoscale-resolving model configuration used to measure performance. Finally, we summarize our conclusions and discuss implications for the future of climate modeling in section 7.

2 Hydrostatic Boussinesq dynamical core equations

Our dycore solves both the Boussinesq equations under the hydrostatic approximation, relevant for large-scale global ocean modeling. The dycore uses a linear free surface and a geopotential vertical coordinate. The implementation of a non-linear free surface and $z^\star$ coordinates is ongoing and should not hamper the performance of the dycore. The prognostic variables are the horizontal velocities, u and v, the sea-surface height elevation η, the conservative temperature T, and the absolute salinity S. A non-linear equation of state relates the buoyancy of seawater b to temperature, salinity, and depth, i.e., $b = \mathcal{F}(T, S, z)$ (Roquet et al., 2015).

For notation convenience we split the three-dimensional velocity vector $\boldsymbol{u}$ into the horizontal component $\boldsymbol{u}_h$ and the vertical component w,

$$\boldsymbol{u} = \underbrace{u\,\hat{\boldsymbol{x}} + v\,\hat{\boldsymbol{y}}}_{\stackrel{\text{def}}{=}\boldsymbol{u}_h} + w\,\hat{\boldsymbol{z}}\,, \tag{1}$$

where $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{z}})$ represents the basis of an orthogonal coordinate system with $\hat{\boldsymbol{z}}$ always pointing in the local upward direction. When at rest, the ocean's sea surface is at $z = 0$. A spatially varying depth at $z = -H(x, y)$, defines the ocean floor.

With the above definitions, the equations for momentum, mass conservation, and sea-surface height elevation are:

$$\partial_t \boldsymbol{u}_h = \underbrace{-(\zeta + f)\hat{\boldsymbol{z}} \times \boldsymbol{u}_h - \boldsymbol{\nabla}\left(p + \tfrac{1}{2}\boldsymbol{u}_h \cdot \boldsymbol{u}_h\right) - w\partial_z \boldsymbol{u}_h}_{\stackrel{\text{def}}{=}\boldsymbol{G}_u} - \partial_z \boldsymbol{\tau} - g\boldsymbol{\nabla}\eta\,, \tag{2}$$

$$\partial_z p = b\,, \tag{3}$$

$$0 = \boldsymbol{\nabla} \cdot \boldsymbol{u}_h + \partial_z w\,, \tag{4}$$

$$\partial_t \eta = w\big|_{z=0}\,, \tag{5}$$

where $f = 2\Omega \sin\phi$ is the Coriolis parameter with Ω the Earth's rotation rate and ϕ the latitude, g is the gravitational acceleration, $\boldsymbol{\nabla} = \hat{\boldsymbol{x}}\partial_x + \hat{\boldsymbol{y}}\partial_y$ is the horizontal gradient, p is the kinematic pressure, $b \stackrel{\text{def}}{=} -g(\rho/\rho_0 - 1)$ is seawater buoyancy relative to a Boussinesq seawater reference density ρ_0, η the free-surface elevation as measured from rest-height $z = 0$, and $\zeta = \hat{\boldsymbol{z}} \cdot (\boldsymbol{\nabla} \times \boldsymbol{u})$ is the vertical component of vorticity. We used the vector identity $\boldsymbol{u}_h \cdot \boldsymbol{\nabla}\boldsymbol{u}_h = \zeta\hat{\boldsymbol{z}} \times \boldsymbol{u}_h + \boldsymbol{\nabla}\left(\tfrac{1}{2}\boldsymbol{u}_h \cdot \boldsymbol{u}_h\right)$ to rewrite the horizontal advection term in (2) in vector-invariant form. The vertical momentum stress is

$$\boldsymbol{\tau} = \begin{cases} \boldsymbol{\tau}_s\,, & \text{at the top surface} \\ -\nu_e \partial_z \boldsymbol{u}_h\,, & \text{in the interior} \\ \boldsymbol{\tau}_b\,, & \text{at the bottom boundary} \end{cases} \tag{6}$$

where $\boldsymbol{\tau}_s$ the surface stress due to winds and $\boldsymbol{\tau}_b = -C_D \|\boldsymbol{u}_h\|\,\boldsymbol{u}_h$ is quadratic bottom drag with coefficient C_D. Vertical mixing of momentum by subgrid turbulence is represented as downgradient diffusion with a turbulent viscosity ν_e.

The vertical velocity is not a prognostic variable; instead it is diagnosed through the continuity equation (4). Using (4) and boundary conditions at the ocean's bottom, we rewrite the free-surface evolution equation (5),

$$\partial_t \eta = -\boldsymbol{\nabla} \cdot \underbrace{\int_{-H}^{0} \boldsymbol{u}_h \,\mathrm{d}z}_{\stackrel{\text{def}}{=}\boldsymbol{U}}\,, \tag{7}$$

where we introduced a new two-dimensional variable, the depth-integrated or 'barotropic' transport $\boldsymbol{U}$. Thus, the evolution of η is complemented by the evolution of the barotropic transport that evolves according to the vertically-integrated horizontal momentum equation:

$$\partial_t \boldsymbol{U} = -gH\boldsymbol{\nabla}\eta + \int_{-H}^{0} \boldsymbol{G}_u \, \mathrm{d}z - \boldsymbol{\tau}_s + \boldsymbol{\tau}_b \,. \tag{8}$$

The ocean dynamics described by (2), (3), (4), (7), and (8) involve two different timescales: a fast timescale that is related to the barotropic flow and the sea-surface height, and a slower timescale that is related to the depth-dependent flow (the "baroclinic" flow). For typical ocean conditions, the barotropic dynamics evolve about 30 times faster than the baroclinic dynamics.

To resolve both the faster barotropic and slower baroclinic timescales we use a split–explicit algorithm (Gadd, 1978; Killworth et al., 1991). The barotropic two-dimensional evolution equations for the sea-surface height and the barotropic transport are advanced using shorter time steps within the longer baroclinic time step that is used for the fully three-dimensional baroclinic dynamics. In particular, all terms grouped as $\boldsymbol{G}_u$ in (2) are assumed to evolve slowly relative to the last term that involves the sea-surface height gradients. This is not formally true for the Coriolis and the nonlinear terms that are characterized by some fast-evolving dynamics, but it is a reasonable approximation when running mesoscale resolving simulations that require time-steps shorter than five minutes.

In conclusion, the hydrostatic ocean model thus comprises of (2), (3), (4), (7), and (8), together with evolution equations for the tracers, which are advected by the total flow (1):

$$\partial_t c = \underbrace{-\boldsymbol{\nabla}\cdot(\boldsymbol{u}_h c) - \partial_z(wc)}_{\stackrel{\mathrm{def}}{=} G_c} - \partial_z J^c \ , \tag{9}$$

where c denotes temperature T, salinity S, or any other tracer. The vertical tracer flux is:

$$J^c = \begin{cases} J^c_s \ , & \text{on the top boundary} \\ -\kappa_e \partial_z c \ , & \text{in the interior} \\ J^c_b \ , & \text{on the bottom boundary} \end{cases} \tag{10}$$

where J^c_s is the flux of c at the ocean surface, while J^c_b is the bottom flux, and the tracer is mixed in the vertical at a rate given by the turbulent diffusivity κ_e.

2.1 Spatial and temporal discretization

We discretize the governing equations in a finite volume framework on an Arakawa staggered C-grid (Arakawa & Lamb, 1977). We employ a second-order spatial discretization for the pressure terms, the continuity equation, the vertical transport, as well as the gradients in (7) and (8). The horizontal transport terms are implemented using a seventh-order weighted essentially non-oscillatory (WENO) scheme. The WENO scheme adapts to local flow and tracer gradients and thus removes the need for explicit stabilizing viscosity or diffusivity. The momentum advection follows the new WENO implementation described by Silvestri et al. (2024); with the difference that the vertical advection term $\partial_z(w\boldsymbol{u}_h)$ is discretized using a second-order centered reconstruction scheme instead of a fifth-order WENO as described in the reference.

Following the split-explicit algorithm described above, we denote the short barotropic step as Δt_S and the long baroclinic time step as Δt_L. Assuming $\Delta t_L = N\Delta t_S$, typically in ocean simulations, $N \approx 30$. In our simulations, we use $N = 50$ substeps and employ the minimal dispersion filter introduced by Shchepetkin and McWilliams (2005) to average barotropic variables over the substeps. The barotropic step Δt_S is calculated as to center the averaging filter at the new baroclinic time step, therefore $N\Delta t_S > \Delta t_L$. The baroclinic dynamics are evolved using a pseudo Adams–Bashforth time-stepping method (formally first order) where the tendency used to evolve velocities and tracers at time step $n+1$ is extrapolated from the previous two time steps as

$$G^{n+1} = \left(\frac{3}{2} + \chi\right) G^n - \left(\frac{1}{2} + \chi\right) G^{n-1} \, . \tag{11}$$

where $\chi = 0.1$. This time-stepping scheme is not state-of-the-art due to the implicit diffusion used to stabilize the nonlinear term through the additional constant χ. Nevertheless, it is a good starting point for GPU execution because it allows explicit calculation of the tendencies and reduces the requirement for memory allocation (see Section 3). However, the same characteristics apply to more sophisticated explicit time-stepping schemes with higher order accuracy, like low-storage Runge-Kutta schemes, which we plan to implement in future work. The barotropic sub-stepping is performed using a Forward–Backward scheme in the following fashion

$$\eta^{m+1} = \eta^m \; - \Delta t_S \boldsymbol{\nabla} \cdot \boldsymbol{U}^m \; , \tag{12}$$

$$\boldsymbol{U}^{m+1} = \boldsymbol{U}^m - \Delta t_S \left(gH \boldsymbol{\nabla} \eta - \int_{-H}^{0} \boldsymbol{G}_u^{n+1} dz - \boldsymbol{\tau}_s^{n+1} + \boldsymbol{\tau}_b^{n+1} \right) \; , \tag{13}$$

where $\boldsymbol{G}_u^{n+1}$, $\boldsymbol{\tau}_s^{n+1}$, and $\boldsymbol{\tau}_b^{n+1}$ — frozen during substepping — are extrapolated using the quasi-Adams-Bashforth scheme shown in eq. (11). As we show in Section 3 and Figure 7, the number of substeps is irrelevant with respect to performance, as the two-dimensional computation of the free surface is extremely lightweight. Therefore, contrary to the baroclinic mode, where a better time-stepping scheme could be implemented, probably leading to a performance improvement, more sophistication in time discretization for the barotropic mode is not warranted on GPUs. Finally, the vertical mixing, which involves large diffusivity terms, is evaluated implicitly column-wise with a backward Euler time-stepping scheme by applying a tri-diagonal solver.

3 GPU-tailored implementation of the hydrostatic Boussinesq equations

The ocean dynamical core we present is implemented in Oceananigans (Ramadhan et al., 2020), an open source library that solves both the hydrostatic and nonhydrostatic Boussinesq form of the incompressible Navier–Stokes equations in Julia (Bezanson et al., 2017). Oceananigans was built from scratch in the Julia language, using a design philosophy rooted in the proven finite-volume principles for ocean dycores pioneered by MITgcm (Marshall et al., 1997). Starting from a clean slate allowed us to adopt implementation practices optimized for GPUs that differ from methodologies prevalent in ocean models optimized for CPUs. We note that the techniques described in this section are not necessarily new with regard to GPU computing. The GPU optimization process, following a standard bottleneck identification and analysis procedure, has been described a number of times for different software and algorithms (e.g., see Micikevicius (2010)). Moreover, many CFD softwares have adopted GPU-specific optimization techniques like those described in this section and obtained efficient execution on GPUs (Costa et al., 2021; Räss et al., 2019; Sætra, 2013). However, we describe here the application of such techniques specifically in the framework of an ocean model.

GPUs excel at executing algorithms that can be highly parallelized, such as computational fluid dynamics. The smallest parallel GPU units, called threads, run concurrently, enabling the simultaneous processing of multiple operations. Threads are organized into groups called thread blocks that can read and write into a shared global memory (DRAM), the primary storage space for GPU variables with a slow input/output access. For efficient management and execution, threads are further grouped into sets of 32, referred to as “warps”. A single scheduling unit manages each warp, adhering to the Single Instruction, Multiple Thread (SIMT) execution model. This model ensures that all threads in a warp execute the same instruction at the same time. Functions executed on GPUs are called “kernels”. Kernels are launched on a "thread-block" grid, with threads that execute in parallel following the SIMT model (NVIDIA Corporation, 2010).

In this section, we describe the implementation details of Oceananigans' hydrostatic algorithm and illustrate how the computational approach makes efficient use of GPU architectures. The algorithm comprises four "macro-areas",

$$\partial_t \boldsymbol{u}_h = \boxed{\boldsymbol{G}_u} - \boxed{\partial_z \boldsymbol{\tau}} - \boxed{g \boldsymbol{\nabla} \eta}\,, \tag{14}$$

$$\boxed{\partial_z p} = b\,, \tag{15}$$

$$0 = \boldsymbol{\nabla} \cdot \boldsymbol{u}_h + \boxed{\partial_z w}\,, \tag{16}$$

$$\boxed{\partial_t \eta} = w\big|_{z=0}\,, \tag{17}$$

$$\partial_t c = \boxed{G_c} - \boxed{\partial_z J^c}\,, \tag{18}$$

where

1. red terms in (14) and (18) refer to the calculation of the tendency of the three-dimensional prognostic variables, including horizontal velocities and tracers;
2. blue terms in (14) and (18) refer to the execution of the implicit vertical diffusion through a backward Euler step, achieved by inverting a tri-diagonal matrix;
3. green terms in (14) and (17) refer to the update of the barotropic velocities and the sea-surface elevation using a barotropic solver, and
4. yellow terms in (15) and (16) refer to the computation of the diagnostic variables, such as vertical velocity, hydrostatic pressure, and diffusivities.

Solving (14)–(18) on GPUs necessitates mapping a kernel onto a parallel thread-block configuration. Notably, a significant portion of computational resources is allocated to calculating the tendency terms. This computation inherently lends itself to parallelization, as each computational cell is independent of others. Consequently, we opt to parallelize the tendency computation using a three-dimensional kernel, with each thread managing the calculations of a single computational cell.

Conversely, implicit vertical diffusion involves inverting a tridiagonal matrix in the vertical direction. Therefore, a more suitable parallelization approach involves launching a two-dimensional kernel, where each thread is responsible for solving the linear system in a computational column. Since the linear system is solved entirely by one individual thread, the computation is effectively serial, allowing the use of fast algorithms developed for serial computation. In our case, we use a direct sweep (the Thomas algorithm) with forward elimination and backward substitution as described in Sakharnykh (2009). Since implicit diffusion operates in the vertical direction, where k (the vertical index) corresponds to the slowest moving index in memory, consecutive threads access consecutive i indices (corresponding to the zonal direction) leading to an improved coalescing of memory access. Barotropic dynamics are inherently two-dimensional, so the barotropic solver requires only two-dimensional kernels where each thread holds one computational cell. Finally, in the GPU implementation of the diagnostic variables' computation, if a kernel necessitates vertical integration (e.g., vertical velocity and hydrostatic pressure), it is implemented as a two-dimensional kernel similar to implicit diffusion. If the computation is inherently three-dimensional (e.g., calculating a local diffusivity), a three-dimensional kernel is launched instead.

3.1 Optimization of the memory footprint

Modern GPU devices pair several thousand floating point units alongside a comparatively limited pool of high-bandwidth memory. An effective strategy for utilizing the GPU's compute resources is to increase the number of grid points assigned to each GPU by minimizing the use of temporary arrays. This approach, common in GPU-based software (Awan & Saeed, 2016; Jakob, 2019), results in a reduction of the dycore's memory footprint but requires weighing trade-offs, especially the higher computational overhead from recalculating quantities that could be precomputed and stored in temporary memory. This tradeoff is dependent on the specific implementation, and each new GPU model should independently assess how much temporary memory to allocate. For example, in CPU-based ocean models, intermediate arrays are often used to store variables like spatially interpolated velocities for calculating advective transport terms, or vertical vorticity used for momentum advection. However, the number of such arrays scales with the number of variables, the terms in the equations being solved, and the dimensions, often dominating the code's memory footprint. Here, we minimize the number of temporary arrays during model time-stepping to optimize GPU memory use, allowing larger problem sets on fewer GPUs — a critical consideration given GPUs' limited high-speed memory.

```julia
@kernel function compute_Gu!(Gu, advection, coriolis,
                             velocities, pressure,
                             closures, diffusivities,
                             args...)

    # `i`, `j`, and `k` are the x, y, and z-thread indices
    # corresponding to CUDA's `threadIdx.x + blockIdx.x * blockDim.x`, etc...
    i, j, k = @index(Global, NTuple)

    @inbounds Gu[i, j, k] = - U_dot_∇u(i, j, k, grid, advection, velocities, args...)
                            - x_f_cross_U(i, j, k, grid, coriolis, velocities, args...)
                            - ∂xᶠᶜᶜ(i, j, k, grid, pressure)
                            - ∂ⱼ_τ₁ⱼ(i, j, k, grid, closures, diffusivities, velocities, args...)

end
```

Figure 2: A code fragment that illustrates the point-wise, functional coding style used in Oceananigans to compute the zonal component of the $\boldsymbol{G}_u$ term in the momentum equation (2). The architecture-agnostic kernel syntax is made possible by the KernelAbstractions.jl library.

In Oceananigans, the tendency for each prognostic variable is calculated in a single kernel, with individual threads computing each grid cell's contribution. This circumvents the need for extra intermediate arrays, as the tendency computation requires only the prognostic and few diagnostic variables. The result is significant kernel fusion, highly beneficial on GPUs (G. Wang et al., 2010), and a reduced memory footprint. Figure 2 illustrates a fragment of Julia code that evaluates the tendency of the u-velocity, i.e., the zonal component of $\boldsymbol{G}_u$, in (2). The code fragment in figure 2 shows how all the tendency computations are performed pointwise without using intermediate variables. Characteristically, a double-precision 1/12th-degree horizontal resolution simulation with a hundred vertical levels requires around 150 GB of memory. Balaji et al. (2017) define *bloat* as the ratio of the total memory footprint to the ideal memory occupied by the prognostic variables. With 5 prognostic variables (u, v, T, S, and η) totaling 25 GB, the excess memory is 125 GB, or an equivalent *bloat* of 5.0 (note that the free surface η is two-dimensional). This value is relatively small compared to the bloat of a typical ocean model, ranging from 10 to 100 (Acosta et al., 2024). A large improvement in memory footprint (and probably performance) would be achieved by switching the computation to single precision. Oceananigans is capable of operating at different precision. However, the implementation is naive, that is, it does not compensate for the effect or the reduced precision in precision-dominated bottlenecks (Prims et al., 2019).

For this reason, we avoid encouraging single precision computations until we have verified and validated the dynamical core with 32-bit floats.

The kernel in figure 2 also showcases that the Julia library KernelAbstractions.jl (Churavy et al., 2024) used in Oceananigans allows us to compose architecture-agnostic kernels that can seamlessly execute on either GPU and CPU platforms within the same code base, similar to kernels written using alternative libraries such as HIP (Gupta & contributors, 2024) or Kokkos (Christian et al., 2021).

3.2 Sparse compute framework

A warp executes one common instruction at a time, so full efficiency is realized when all 32 threads of a warp follow the same execution path. If threads of a warp diverge due to a data-dependent conditional branch, the warp executes all the paths entirely, disabling threads that are not on that path. This performance loss, unique to GPUs, is termed *branch divergence*. Branch divergence is typical of GPU-based solvers that include stochastic elements, for example, Monte Carlo solvers characterized by while loops with stopping criteria based on sampling of random numbers (Silvestri & Pecnik, 2019). However, given the deterministic nature of fluid dynamics computation, branch divergence is uncommon in GPU-based fluid dynamics software as, generally, divergent tasks are limited in size ans can be reduced to divergence-free implementations (Tran et al., 2017). However, the presence of boundaries and boundary conditions requires special care for boundary-adjacent grid points that can potentially lead to branch divergence.

In our dynamical core, branch divergence can arise from two primary sources. Firstly, it can stem from the utilization of high-order numerical schemes for advection: the stencil reconstruction of the high-order numerical scheme is constrained to lower-order reconstruction near boundaries. Consequently, threads that manage cells near to land boundaries end up having to perform different computational tasks than the cells in the ocean's interior. This potentially results in divergent executions within a warp. We have chosen to avoid branching by performing the same computation in each thread. This increases the compute time, hence a better separation of boundary versus interior threads ought to be explored to improve code performance.

The second possible source of branch divergence arises from the representation of bathymetry. Oceananigans uses a structured mesh, where "land" cells below bathymetry are masked and the velocity components normal to the solid interfaces are set to zero. This approach, depicted in the algorithm on the top panel of figure 3, is commonly employed in structured ocean models. However, performance dramatically decreases on GPUs, where both branches are executed in the event of a diverging conditional. In practical terms, this entails launching threads for "land" cells that do not actively engage in the computation but unnecessarily occupy resources as they wait for the threads performing the computations in "ocean" cells. To address this issue, we implemented a "sparse compute" framework inspired by the approach with the same name described in Sætra (2013). Active cells, representing ocean cells participating in the computation, are identified and mapped during a preprocessing step. The map is stored in a one-dimensional list of active indices. Subsequently, the kernels are launched with a number of threads equivalent to the total number of active cells in the map. Within these kernels, the three-dimensional index is retrieved from the precomputed map, allowing the computation to proceed as usual. An example of a "sparse compute" kernel is shown on the bottom panel in figure 3. Note that, with this approach, we are trading branch divergence with possibly uncoalesced memory access, so the success of this framework depends on the ratio of "land" to "ocean" cells. By adopting this methodology, particularly in simulations like the global ocean where 42% of the grid cells are immersed, we achieved a notable speedup of up to 2×.

Algorithm 1: Divergent kernel launch

```
# Launch the kernel on a GPU grid mapped to the physical grid
worksize = (Nx, Ny, Nz) # the 3D size of the grid
launch!(calculate_tendencies_kernel, worksize)

@kernel function _tendencies_kernel!(model, grid)
   i, j, k = @index(Global, NTuple)
   if !immersed_cell(i, j, k, grid)
        calculate_local_tendencies(i, j, k, grid, model)
   end
end
```

Algorithm 2: GPU-optimized kernel launch

```
# active_cells is a list of active indices (i, j, k)
worksize = length(active_cells)
launch!(calculate_tendencies_kernel, worksize)

@kernel function _tendencies_kernel!(model, grid, active_cells)
    idx = @index(Global, Linear)
    i, j, k = active_cells[idx]
    calculate_local_tendencies(i, j, k, grid, model)
end
```

Figure 3: Example of domain loop using a divergent kernel (top) where non-active "land" cells stall while waiting for active "ocean" cells, and a GPU-optimized "sparse compute" kernel (bottom).

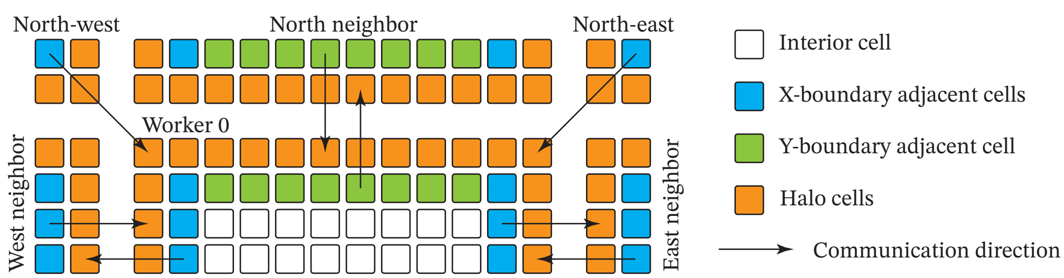

Figure 4: A schematic depicting the communication and computation layout of the parallel implementation in a one-halo configuration. The tendencies in the interior cells (white) are computed concurrently with the communication in the halo cells (orange). When the communication finishes (typically before the completion of the interior computations), two different kernels computing the tendencies in the boundary-adjacent cells are executed.

3.3 Scalable parallelization

GPU execution of parallelizable tasks typically outperforms CPU execution due to the GPU's inherent parallel processing capabilities. However, inefficient parallelization across multiple GPUs can lead to communication becoming the main bottleneck of simulation (Wei et al., 2023; Häfner et al., 2021). Consequently, achieving scalability on numerous GPUs poses greater challenges compared to CPU architectures and requires careful implementation of algorithmic logic to mitigate performance bottlenecks effectively.

In ocean models, it is common to allocate additional cells on the boundaries of the domain, referred to as "halo" or "ghost" cells, which hold the results of neighboring processors. These results are typically communicated through a message-passing communication step (Marshall et al., 1997). In Oceananigans, we have implemented communication–computation overlap, hiding the cost of communicating halo regions behind kernel computations. Communication–computation overlap for the three-dimensional baroclinic variables uses the same straightforward approach found in many high-performance GPU finite volume libraries, for example,

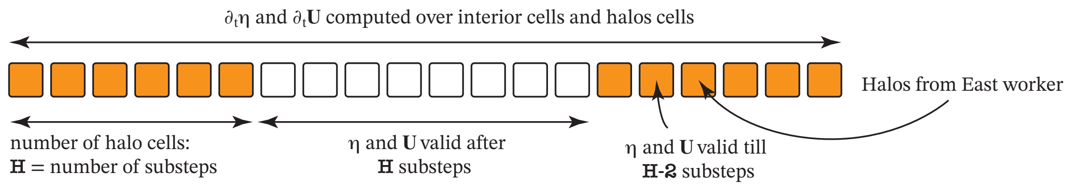

Figure 5: A schematic depicting the computation layout of the parallel barotropic solver in one dimension. The η and $\boldsymbol{U}$ equations are solved on the entire domain including halos, with the number of halo cells equal to the number of subcycles (barotropic time steps). After advancing through the subcycles, the values of η and $\boldsymbol{U}$ are valid only in the domain's interior.

Räss et al. (2019): splitting the large tendency computation into boundary-dependent and boundary-independent regions.

A schematic of this process is shown in figure 4. This figure shows a two-dimensional domain split into four different regions (the southern boundary is not shown). The orange cells represent the "halo" cells. The interior domain is divided into three different kernels. White cells represent the "inner" region that is boundary-independent. The tendency in these cells can be computed while communication among GPUs is in progress. Boundary-dependent cells are colored green and blue. The kernels for computing tendencies in these regions, which depend on the halo cells, are launched after communication is completed. Note that figure 4 shows the simple case of second-order numerics where only one halo cell is required; for higher-order spatial discretizations the boundary-dependent regions are larger and the inner region decreases in size.

As discussed in section 2, in hydrostatic ocean models with a free surface, the vertically-averaged, two-dimensional "barotropic" flow represents dynamics that evolve an order of magnitude faster than the three-dimensional "baroclinic" component. Therefore, the special "barotropic solver", which is typically computationally cheap given that the problem is two-dimensional, is communication-intensive since the different cores (or GPUs, in our case) need to communicate at each substep. It is precisely because of this communication overhead that the barotropic mode in ocean models — whether using implicit or split-explicit solvers — constitutes a major bottleneck that accounts for between 40% (Häfner et al., 2021; Kang et al., 2021) and 60% (P. Wang et al., 2021; Hu et al., 2013) of the cost of a typical IPCC-class ocean simulation.

To improve the scalability, we have adopted an optimization for the parallel barotropic solver tailored to GPUs, which might also increase efficiency in CPU-based ocean models. This optimization is particularly effective for memory-efficient code that allows many points on each GPU, in our case around 10^8 (see section 3.1). It involves trading a slight increase in computation for decreased communication latency by capitalizing on the two-dimensionality of the barotropic mode. In practice, we expand the horizontal extent of the halo region of barotropic variables to match the number of explicit substeps (typically between 30 and 50) and convert halo cells to active cells. This leads to an increase in the cost of barotropic computation because barotropic tendencies also computed in halo regions. However, since the barotropic solver is two-dimensional, the cost of this extra computation is negligible. On the other hand, by performing this optimization (as illustrated in figure 5) communication is necessary only once per baroclinic time step rather than every subcycle, thereby decreasing the communication frequency by 30 to 50 times. In addition, since vertical diffusion and the barotropic step are commutative, we can communicate the halos of the barotropic variables asynchronously while performing the implicit vertical diffusion step. As a result of the sparsity of communication enabled by our barotropic solver implementation, all communication operations can overlap with computational kernels. Consequently, for typical

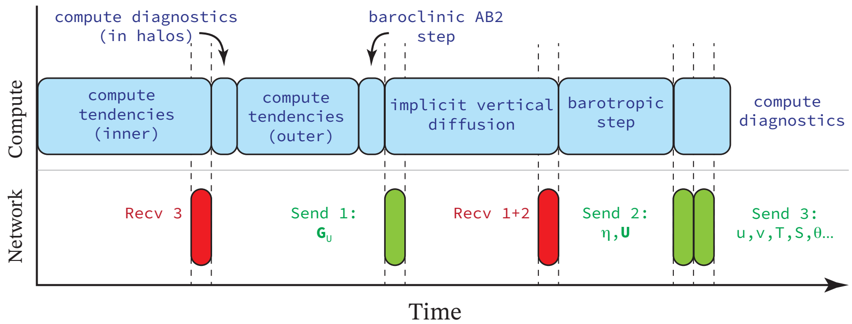

Figure 6: Schematic depicting the algorithmic flow and the communication–computation overlap.

ocean simulation domains, the cost of the barotropic solver diminishes to less than 10% of the total cost of a time step, as demonstrated in section 5.

Figure 6 outlines the logic of Oceananigans' hydrostatic algorithm, highlighting two main advancements compared to a classical CPU ocean model implementation: (1) dividing large tendency kernels and auxiliary computations into "inner" and "outer" kernels–typically not performed in CPU codebases but necessary for GPU computation; (2) concealing the communication of barotropic variables behind the implicit vertical diffusion by enlarging the barotropic halo regions to match the number of subcycles.

4 Model configurations for performance testing

We configured our ocean model in two setups to test its performance: a quasi-realistic near-global ocean configuration and a more simplified Double Drake configuration described by Ferreira et al. (2010). The Double Drake configuration consists of a 3 km-deep, flat-bottom ocean covering the full planet except for two one-degree wide walls extending from the northernmost latitude to 35° south and separated by 90° degrees in longitude. This configuration provides a less ambiguous test for weak scaling than the quasi-realistic configuration because the topography does not change when increasing problem size and number of GPUs.

Both configurations use a latitude-longitude horizontal mesh extending from 75°S to 75°N, with a z-coordinate vertical discretization using 100 vertical layers with thickness ranging from 2.5 m at the surface to 200 m at the bottom. Note that given the different depth of the setups, the maximum grid size is slightly different between the two. The buoyancy is calculated from T and S using a polynomial approximation to the TEOS-10 equation of state (Roquet et al., 2015). Vertical mixing by unresolved small-scale turbulence is parameterized through a vertical diffusivity and viscosity which are nonlinear functions of the Richardson number (see Appendix A). Horizontal mixing of momentum and tracers is implicit through the WENO implementation of the advective terms (Silvestri et al., 2024); no explicit lateral mixing is introduced. There is no sea ice component.

The Double Drake configuration is forced with a zonal wind stress which depends only on latitude and mimics the actual zonal-wind stress acting on the Earth's oceans. The buoyancy forcing is through a surface relaxation to a parabolic function of latitude. This setup is only used for performance and stability testing, so it has been integrated for only 1 year. The configuration was run at different horizontal resolutions spanning 1/6° to 1/168°.

The quasi-realistic configuration uses an ocean bathymetry interpolated from ETOPO1. The surface forcing is taken from the 1995 repeat-year daily fluxes interpolated from the ECCO2 CS510 product (Menemenlis et al., 2008). The wind stress is applied as a mechanical input in the surface layer. The temperature and salinity forcing are imposed as the sum of the interpolated ECCO2 heat and salinity fluxes plus a restoring to the three-day averaged surface temperature and salinity ECCO2 fields with 90 and 45 meters per year piston velocities, respectively. Initial conditions for temperature and salinity are generated by interpolating the ECCO2 January, 1st, 1995 temperature and salinity fields onto the model grid. The velocity and the free surface field are initially at rest. To assess performance the model was run for 1000 time steps with varying horizontal resolution: $1/4°$, $1/8°$, $1/12°$, $1/48°$, and $1/96°$.

To complement the performance results shown in section 5, the $1/12°$ resolution was integrated for a total of 20 years to showcase a mesoscale resolving solution. The baroclinic time step starts at 10 seconds during spinup and is progressively increased reaching 270 seconds by the second month of simulation. We verified that 20 years is sufficient time for the upper ocean velocity to adjust to the density field and for mesoscale processes to reach quasi-equilibrium (Iovino et al., 2016; Ringler et al., 2013). In section 6 we show that the eddy statistics in this simulation compare favorably with observations and provide support for the eddy-rich capabilities of our ocean model.

In addition to the $1/12°$ experiment, we have evolved a higher-resolution version ($1/48°$ degree in the horizontal) for one simulated year, to demonstrate that the model can be run stably at even higher submesoscale resolving resolutions. Fig. 1 shows snapshots of vertical vorticity for the $1/12°$ and the $1/48°$ degree setups after one year of integration. The $1/12°$ model has a horizontal spacing $\Delta \approx 8$ km which appears sufficient to capture the dominant mesoscale eddies, visible as anomalously positive or negative ζ patches with characteristic scales of 50-100 km in lateral extent. The $1/48°$ model has a horizontal spacing of $\Delta \approx 2$ km. At this higher resolution, a rich sub-mesoscale eddy field fills the solution.

Finally, we run a simulation at a $1°$ resolution configured and forced like the $1/12°$ resolution simulation and also run for a full 20 years. The only significant difference is that this simulation uses a 5th order WENO scheme for both tracers and momentum with an additional biharmonic dissipation with a grid-size dependent viscosity of the form $\nu_4 = \Delta^4/\tau_\nu$, where τ_ν is a timescale equal to 15 days and Δ the grid size. The baroclinic time step, limited by vertical advection, is set to 900 seconds. The $1°$ resolution simulation is too coarse to generate any eddies and is used as a comparison to illustrate the impact of the mesoscale eddy field on the large-scale ocean structure in the $1/12°$ resolution simulation.

5 Performance results

In this section, we report the performance of the dycore using the various model setups described above. The performance results shown in this section pertain only the dynamical core and not the I/O that will depend on the particular diagnostics required by users. Where not explicitly mentioned, the results are obtained on the NERSC supercomputer Perlmutter. Perlmutter is an HPE (Hewlett Packard Enterprise) Cray EX supercomputer that hosts four A100 GPUs with 40GB per node, linked through an NVLink3 interconnect.

Figure 7 displays the output of the Nvidia profiler `nsys` for the $1/48°$ quasi-realistic setup on 256 A100 GPUs. This figure illustrates the actual relative time-step execution corresponding to the schematic depicted in figure 6, where the blue boxes delineate the timeline of kernels on a single GPU. Within figure 7, the pertinent algorithmic macro-areas are highlighted by black boxes, along with the send operations corresponding to the schematic shown in figure 6. Receive operations are not shown in the profiles. Notably, despite utilizing a large number of GPUs (256), the communication overhead remains minimal, highlighting the parallel scalability of the dynamical core.

A summary of the information shown in figure 7 is presented for three different configurations in figure 8. Here, we illustrate the percentage of time spent in the execution of the various kernels for the quasi-realistic setup at $1/4°$ on 4 GPUs, $1/12°$ on 64 GPUs, and $1/48°$ on 256 GPUs. Consistent with previous results, the majority of computational

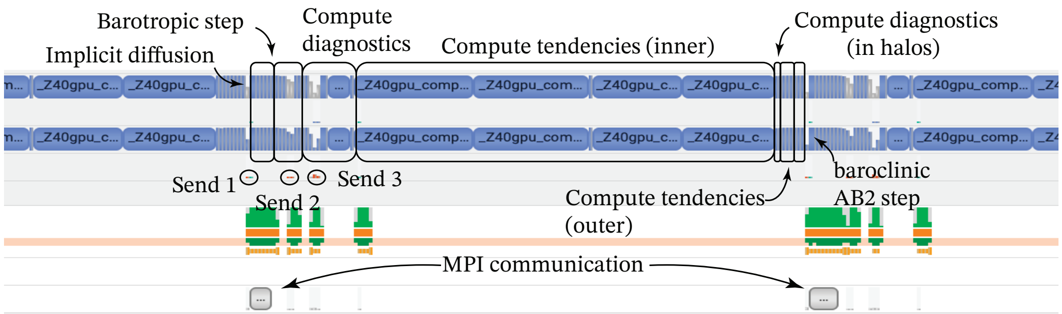

Figure 7: Algorithmic flow and communication–computation overlap in the quasi-realistic ocean setup at 1/48° horizontal resolution and 100 vertical levels on 256 GPUs generated using the Nsight system profiler. The receive operations are not shown.

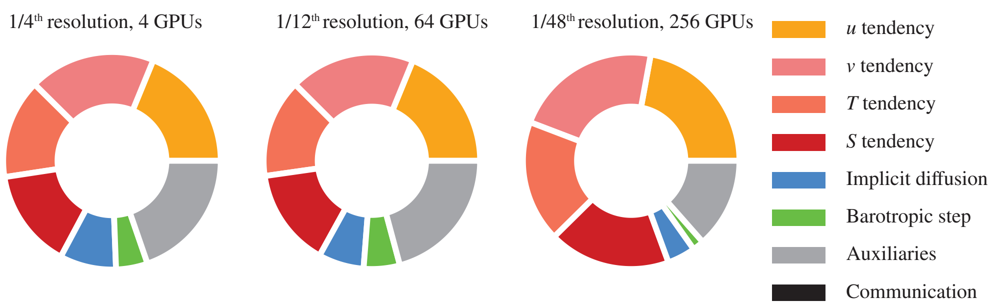

Figure 8: Share of time spent in different kernels for the three quasi-realistic ocean configurations.

resources are consumed by the tendency calculations, with the velocity kernels (u and v) occupying a slightly larger share of resources compared to the tracer kernels. Notably, owing to the implementation of the wide-halo barotropic solver, the barotropic step accounts only for a minimal proportion of resources in all configurations, and communication is completely overlapped with computation.

Figure 9 depicts the performance of the time stepping kernels gathered using Nvidia's compute profiler (`ncu`) in the Double Drake setup at 1/3° horizontal resolution on a single Titan V GPU. Performance is evaluated in terms of TFLOP per second against the arithmetic intensity of the kernel, which quantifies how many FLOPs per memory-retrieved byte are executed in the kernel. When the arithmetic intensity is insufficiently high, the kernel lacks the computational workload necessary to conceal the large latency of memory fetches, rendering it "memory-bound". Conversely, if the arithmetic intensity is high, warps may stall due to instruction latency, leading to the kernel being categorized as "compute-bound".

The small implicit vertical diffusion and barotropic evolution kernels are relatively simple, lacking sufficient arithmetic intensity to effectively mask memory fetch latency. Consequently, these small kernels are memory-bound, limited by the bandwidth of global memory fetch. In contrast, the large tendency kernels, that utilize a high-order WENO reconstruction, demand a significant number of FLOPs per retrieved byte, effectively moving the tendency kernels within the "compute-bound" region of the roofline model.

As a comparison, we showcase the performance of the same kernels but using a simple centered second-order advection instead of the WENO scheme. Although the FLOPs/byte increase tenfold (or more) with WENO advection, the TFLOPs/s increase only by a factor of 2, with a maximum of 2.6 TFLOP/s for the tracer kernels. Therefore, while the use of a

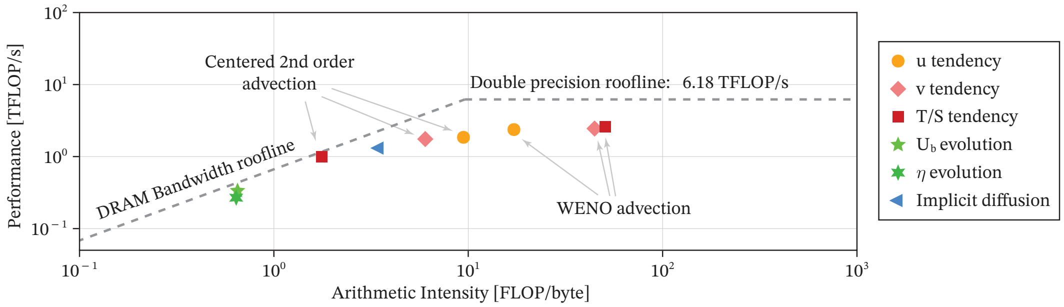

Figure 9: Performance metrics for the relevant GPU kernels in the Double Drake configuration at 1/3° horizontal resolution and 100 vertical levels. The plot shows the speed of the time-stepping kernels measured in TFLOP/s against the arithmetic intensity, i.e. the number of operations per byte in the kernel. The data was gathered on a single Titan V GPU using the Nsight compute profiler (`ncu`). The large tendency kernels (using a high-order WENO scheme) are compared to benchmarks that use second-order centered advection.

WENO reconstruction scheme effectively masks memory fetch latency due to its compute-intensive nature, the kernels fall short of achieving the Titan V GPU's theoretical peak performance of 6.18 TFLOP/s. We suspect this limitation stems from the exceedingly high register pressure of the large tendency kernels (255 registers for the u and v kernels and 180 registers for the tracers) caused by the WENO advection scheme. This pressure restricts GPU occupancy to a mere 11%, eventually leading to the spillover of the register into high-latency local memory. This shows that further optimization to alleviate the register pressure caused by WENO and permit a larger concurrent execution of parallel warps within a streaming multiprocessor could potentially lead to a significant boost in performance (Singh et al., 2018).

5.1 Scaling performance

The scaling of Oceananigans' dycore is illustrated in Figure 10 for the quasi-realistic ocean setup and in Figure 11 for the Double Drake setup. While figure 10 showcases *strong* scaling of the code, which consists in increasing the resources for a fixed problem size, figure 11 showcases *weak* scaling, which involves increasing the resources alongside the problem while maintaining a fixed problem size per GPU. The strong scaling (fixed problem size) is tested using the quasi-realistic setup. For testing the weak scaling efficiency we opted to utilize the Double Drake setup since adapting a quasi-realistic ocean setup to different resolutions is more challenging (requiring interpolation of bathymetry, initial conditions, fluxes, etc...).

Figure 10 shows that the strong scaling of the dycore exhibits nearly ideal behavior up to four times the number of GPUs. This suggests that we could exploit the memory leanness of Oceananigans (see section 3.1) by sacrificing a portion of the memory to accelerate computation by storing intermediate results. The strong scaling efficiency eventually declines to about 70% for sixteen times the number of GPUs. It is important to note that this decrease in efficiency is not due to an increase in communication, as communication is consistently overlapped with computation (see figure 7). Rather, the decline in efficiency stems from poor load balancing when scaling the number of workers. Since we employ a sparse compute framework, a structured partitioning of the domain results in some GPUs having more active cells to compute than others, leading to inadequate load balancing. Effectively addressing load balancing within this sparse compute framework is the subject of ongoing development. In general, we achieve an approximate speed of about 75 simulated years per wall-clock day (SYPD) for a quarter-degree ocean simulation on sixteen A100 GPUs, 10 SYPD for a 1/12° ocean simulation on sixty-four GPUs, and over 1 SYPD for a 1/48° ocean simulation on 512 GPUs.

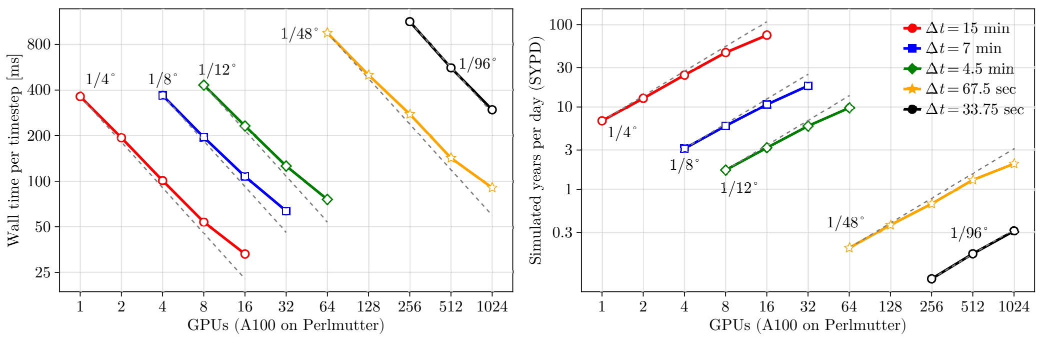

Figure 10: Strong scaling of the quasi-realistic ocean setup in double precision. Different lines show the performance with the number of GPUs for the quasi-realistic setup at 1/4°, 1/8°, 1/12°, 1/48°, and 1/96° horizontal resolution and 100 vertical levels. The simulated years per day are calculated using the time step size shown in the legend on the right-hand side. All results are averaged over 1000 time steps.

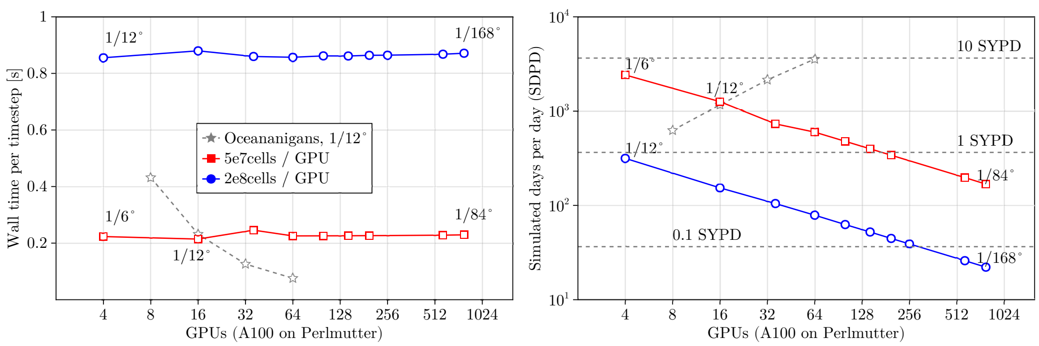

Figure 11: Weak scaling of the "Double Drake" setup in double precision. Each GPU holds a grid equivalent to a 1/12° or 1/6° horizontal resolution and 100 vertical layers. The weak scaling is performed up to a horizontal resolution of 1/168° degree (∼488 m resolution) where we achieve 15 simulated days per wall clock day (1 year in roughly 25 days). The stars mark the strong-scaling performance of the quasi-realistic ocean setup at 1/12° degree resolution as shown in figure 10. All results are averaged over 1000 time steps.

Finally, figure 11 shows the weak scaling capability of Oceananigans' dynamical core in the Double Drake setup, that is, increasing the number of GPUs along with the problem size so that each GPU always handles the same degrees of freedom. We have tested 50 million and 200 million cells per GPU up to a resolution of 1/168° and 1/84° (with 100 vertical levels) on 1 to 192 computational nodes (4 to 768 GPUs). To contextualize the results, the stars show the strong scaling of the mesoscale resolving 1/12° resolution quasi-realistic ocean setup (the same results shown in the previous figure). Given the efficient masking of halo passing and the complete lack of a global communication step, the weak scaling efficiency is ideal in all the investigated configurations.

6 Solutions of the near-global ocean configuration

This section presents some solutions for the quasi-realistic configuration at 1/12° integrated for 20 years. Our goal is to demonstrate that the model can accurately capture the basic features of the global ocean circulation, especially the global ocean mesoscale eddy field in a high-resolution simulation. These tests are not intended to represent state-of-the-art

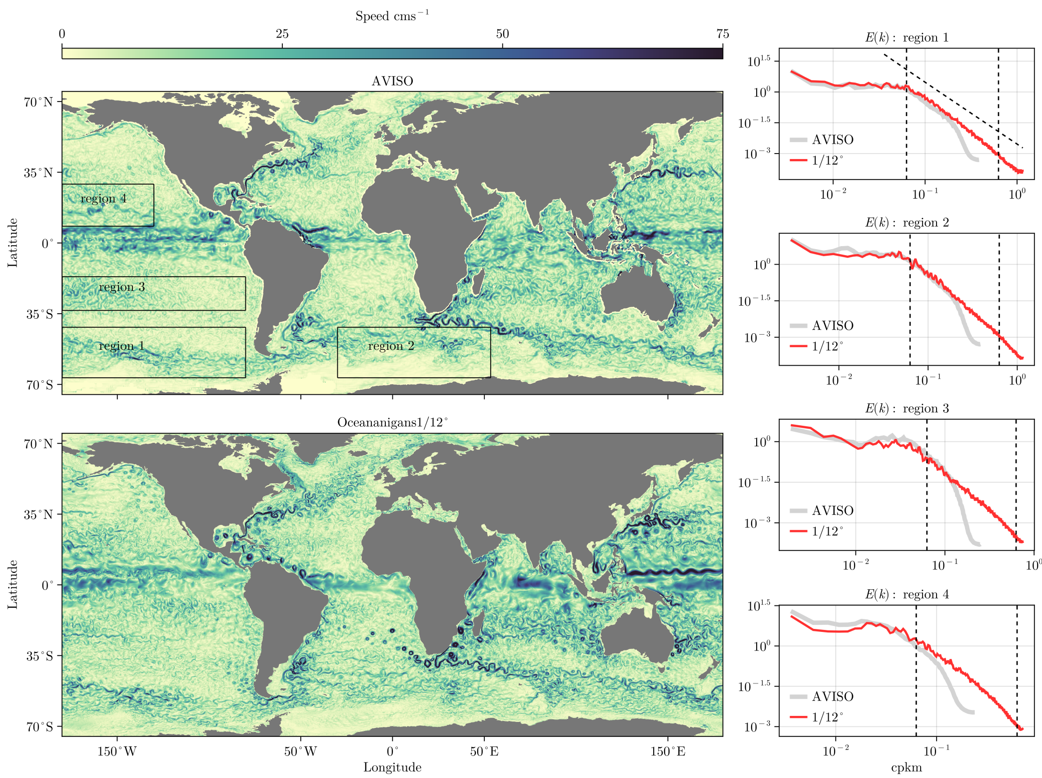

Figure 12: Snapshots of surface speed for the 1/12° model (bottom left) on January 1st compared to the AVISO dataset (top left) on the 30th of December. The plots on the right compare the surface kinetic energy spectra of the modeled solution averaged over the last ten years of evolution (red lines) and the AVISO data in the same region (grey line).

ocean solutions that would require addressing several deficiencies: too short of an integration time for the solution to fully equilibrate, absence of sea ice and an Arctic ocean, simplified surface forcing, and basic parameterization for vertical mixing. Our objective is instead to demonstrate the model skill in generating a realistic mesoscale eddy field; more metrics showing the time-evolution of this configuration are presented in Appendix B.

The two left panels in figure 12 compare surface velocity field snapshots from the simulation and the AVISO (AVISO+, n.d.) satellite-based estimate. The simulation captures the location and magnitude of the most energetic currents. A more quantitative comparison is offered on the right of panels of the figure which show the surface kinetic energy spectra corresponding to the regions highlighted as rectangular boxes in the left panel. The two vertical dashed lines bracket the typical mesoscale length-scale range: 10-100 km. The diagonal dashed line in the top plot shows the expected k^{-3} scaling for kinetic energy spectra in this range of scales (Charney, 1971) (k being the total horizontal wavenumber) .The simulated and AVISO spectra do match very well on the whole range of scales down to the wavenumbers where the AVISO spectra drop off rapidly due to the limited satellite resolution. At even larger wavenumbers, the simulated spectra continue to follow the k^{-3} scaling building confidence that the mesoscale field is well resolved down to the smallest resolved scales.

While the overall pattern and magnitude of surface velocity compare well between simulation and AVISO observations, several differences can be noticed. Both the Gulf Stream and the Kuroshio current deviate southward from the latitudes observed in the altimetry. The Agulhas rings also show some noteworthy deviations from observations. They do shed

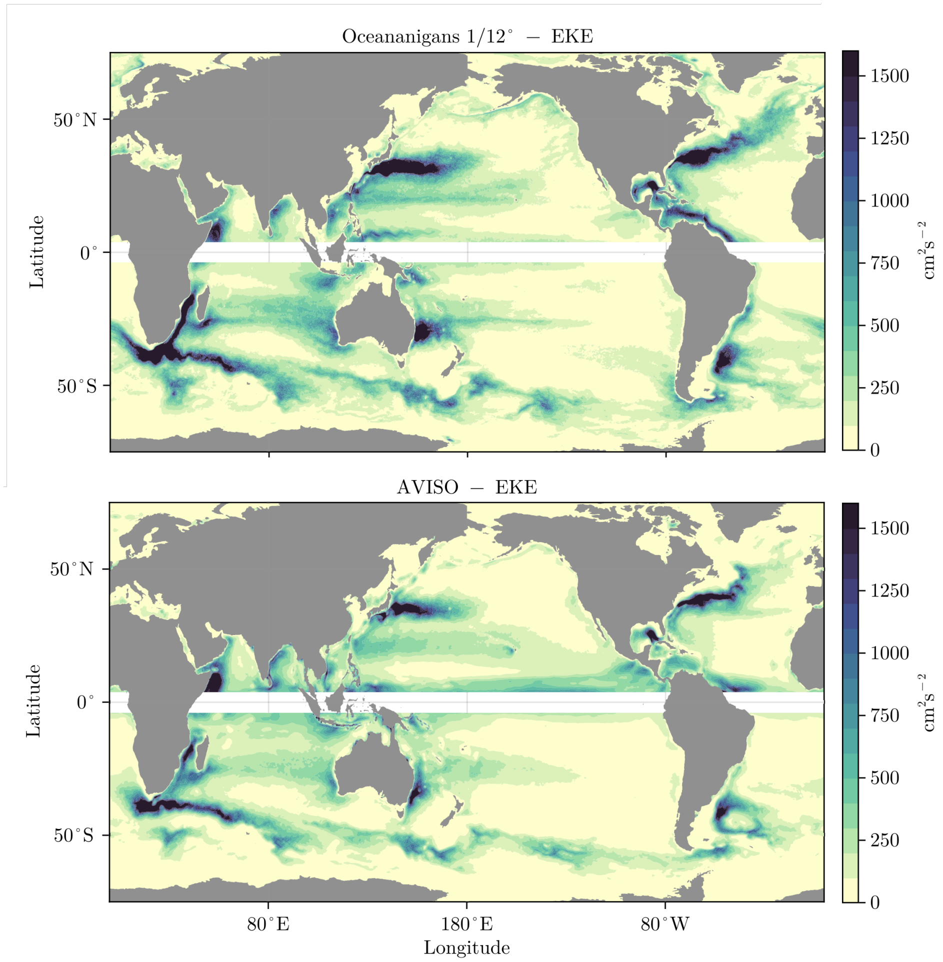

Figure 13: Mean eddy kinetic energy from the 1/12° quasi-realistic simulation averaged over the last 10 years of evolution (left) and from AVISO climatology, averaged over the year 2015 (right)

from South Africa at a frequency comparable to observations and do not all follow a common path, as seen frequently in eddy-resolving models (McClean et al., 2011; Ringler et al., 2013). But, unlike in observations where the rings dissipate early off the coasts of South Africa, in the model, they remain highly energetic and coherent until reaching the coasts of South America. Similarly, the simulated rings that shed off the North Brazilian current reach up to Gulf of Mexico, interacting with the Loop current. No such energetic eddies can be seen in AVISO.

Figure 13 shows the eddy kinetic energy averaged over the last ten years of evolution in the 1/12° model (top) compared to the eddy kinetic energy calculated from the AVISO dataset averaged over thirty years (bottom). Values above 1600 $\mathrm{cm^2\,s^{-2}}$ are saturated. The figure confirms that the numerical model captures the geographical distribution and magnitude of mesoscale variability, which dominates the eddy kinetic energy, not just in a snapshot but also in the time average. The kinetic energy of the mesoscale eddy field in the Southern Ocean seems to be particularly well captured by the model. Differences between the simulation and observations are consistent with those highlighted in the snapshots of figure 12. The model's propensity to sustain longer-lived coherent structures results in elevated eddy kinetic energy along the tracks of the Agulhas rings as well as along the northeastern coast of South America, which are significantly less energetic in the observations. The persistence of mesoscale features is also responsible for the larger spread of high kinetic energy around

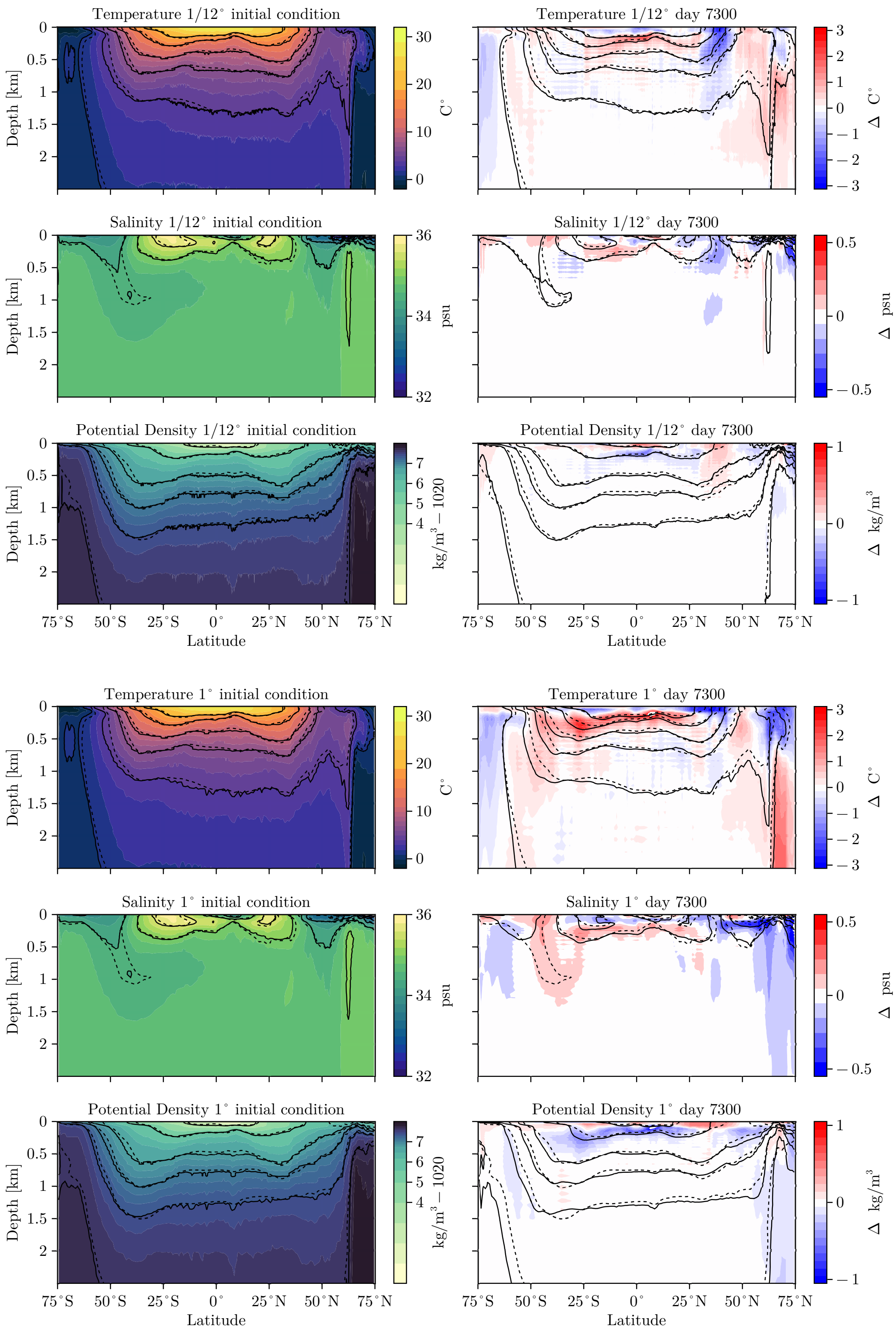

Figure 14: Zonally averaged internal structure on January 1st in the first 2500 meters compared to the EN3 climatological dataset (Ingleby & Huddleston, 2007). The left panels show the initial condition (contour and solid lines) compared to the EN3 data (dashed lines). The right panels show the internal structure after 20 years of evolution (solid) lines compared to the EN3 dataset (dashed lines) superimposed to a contour that illustrates the drift from the initial conditions to the final state (colored contour). The Mediterranean, Caspian, and Black Sea were removed from the dataset before zonally averaging.

the main western boundary currents (Gulf Stream, Kuroshio current, and East Australian current) in the simulation than in the satellite observations.

The tendency of generating spuriously persistent coherent eddies is not unique to our model and has been documented in other eddy-resolving, ocean-only models (Ringler et al., 2013). It is likely that this bias is due by the lack of eddy damping associated with atmosphere-ocean feedbacks. In our simulations, the wind stress is proportional to the atmospheric wind velocity only, rather than the difference between atmosphere and ocean velocities, which results in a damping of the eddy field (e.g., Ferrari and Wunsch (2009)). Indeed preliminary testing using realistic forcing based on bulk formulae and relative atmosphere-ocean velocities resulted in simulations with less persistent coherent structures.

We argued in the introduction that the mesoscale eddy field plays an important role in setting the ocean mean state. To illustrate this point, we now compare the ocean mean state from the quasi-realistic $1/12^{\circ}$ setup, which resolves well the mesoscale eddy field, with that simulated with a 1° setup, which does neither resolve nor parameterize the mesoscale eddy field. Both simulations are run for 20 years. Figure 14 plots the zonally-averaged temperature, salinity, and potential density on January 1st, from both simulations juxtaposed to the EN3 (Ingleby & Huddleston, 2007) climatology for January 1996. The EN3 potential density is derived from temperature and salinity climatology using the same equation of state employed in our dynamical core. The left panels show the initial conditions for the model simulations (colored contour and solid lines) compared to EN3 climatology (dashed lines), while the right panels compare the solution after twenty years of evolution (solid lines) to the EN3 climatology (dashed lines) superposed to the drift between the initial condition and the final state (colored contours). The first three rows show the zonal maps of temperature, salinity, and potential density for the $1/12^{\circ}$ configuration, while the last three show the same results for the 1° setup.

The zonally-averaged profiles of temperature, salinity, and potential density exhibit notably less drift in the $1/12^{\circ}$ configuration compared to the 1° counterpart at all latitudes and depths. This is especially true in the Southern Ocean where mesoscale eddies play a key role in maintaining the stratification. while the isopycnals display little drift in the $1/12^{\circ}$ simulation, in the 1° simulation the stratification decreases significantly from initial to final state. The impact of the eddies is also evident in the mid-latitude thermoclines which become significantly hotter and saltier in the 1° simulation in the absence of the eddies; mesoscale eddies are generated through baroclinic instability which acts to increase the ocean stratification and resist the pumping of heat and salt into the ocean interior.

Poleward of 50°N, both the $1/12^{\circ}$ and the 1° solution depart significantly from the EN3 climatology. The discrepancies are already present in the initial conditions but increase over the following 20 years. We suspect that these discrepancies stem from two main reasons: the absence of a sea ice model and the artificial northern boundary at 75° that ignores the exchange of heat and salt with the Arctic. (The latter is less of a problem in the southern hemisphere where practically the entire Southern Ocean is represented.) That said, even at 50°N, the role of the eddies is reflected in shallower isopycnal slopes for the high-resolution eddying setup when compared to the 1° configuration.

7 Summary and conclusions

We have presented the details of a new GPU-based ocean dynamical core that achieves 10 SYPD at 8 km-resolution using 64 A100 GPUs, equivalent to 16 computational nodes in current state-of-the-art supercomputers such as Perlmutter or Frontier. These resources are similar to (or lower than) the typical resource requirements of state-of-the-art CPU-based ocean models used in climate projections at much *coarser* resolutions of, e.g., 25- to 50 km-resolution, requiring from 10 to 300 computational nodes (Acosta et al., 2024). At these coarser resolutions, ocean models have to rely on parameterizing ocean mesoscale turbulence.

We have demonstrated that the computational efficiency of GPUs can be leveraged to develop climate models that meet time-to-solution requirements for climate projections that, with a lateral spacing below 10 km, do not require mesoscale turbulence parameterizations.

We also note the excellent multiple-GPU scaling yields 1 SYPD at 2 km resolution on 512 GPUs (128 computational nodes on Perlmutter). This paves the way for decadal ocean-only simulations at "submesoscale" resolution, of great importance in the modulation of air–sea fluxes and biological productivity — see J. Taylor and Thompson (2023) — and which is the focus of new satellite platforms (Morrow et al., 2019; Donlon et al., 2012). Sub-kilometer global simulations are also possible (albeit with a large number of GPUs) to study the impacts of sub-mesoscale small-scale ocean turbulence on the global circulation and climate.

We achieved this step-change performance by coding the algorithm from scratch designed specifically for GPUs, including key ocean-model-specific innovations. Both the model structure and numerical algorithm take advantage of the many parallel cores provided by GPUs, while being mindful of the limited access of GPUs to high-bandwidth memory. The algorithm we implemented is independent of the programming language and similar performance could likely be achieved using any other language that allows writing GPU kernels. Examples include CUDA (both C and Fortran versions), HIP and Kokkos. Progress in JIT languages like JAX might also allow achieving similar performance to what we presented in the manuscript with the added benefit of obtaining an automatically differentiable model. Starting from a clean slate, made it easier to consider every algorithmic choice and achieve the remarkable GPU performance reported here. However, we believe it would be possible to achieve similar GPU performance by "translating" an existing CPU-based ocean model while being mindful of the "recipes" described here. These can be broadly summarized as: *(i)* adjust the thread-block grid to the particular algorithmic choice, *(ii)* fuse small computations into one kernel wherever possible, *(iii)* ensure that GPU resources do not idle, and *(iv)* hide communication latency behind computation. If a similar strategy is implemented in other models, future climate model projections could potentially use 10 km-resolution ocean models—perhaps leading to a step-change in the accuracy of climate projections.

In the work described here, we focused on algorithms that can achieve excellent single GPU execution and scaling on multiple GPUs. In particular, we used a finite volume design philosophy such as the one of the MITgcm (Marshall et al., 1997). Different discretization choices, such as the Arbitrary Lagrangian-Eulerian vertical coordinates (Griffies et al., 2020) used to reduce spurious mixing in ocean models, may present greater challenges for efficient GPU implementation. Others, like Discontinuous Galerkin methods (Sridhar et al., 2022; Souza et al., 2023), have shown to be potentially even more suitable for GPU architectures. Finally, one important caveat is that, presently, our ocean model does not include additional components such as representations for sea ice and biogeochemistry. These components would require additional computation and memory storage, resulting in possible performance bottlenecks. While addressing these challenges is a future goal, we believe that the results described here make a strong case for pursuing the benefits of ocean modeling on GPUs.

Appendix A Parameterization for vertical mixing by convective, shear, and background small-scale turbulence

We use a parameterization based on convective adjustment and a stably-stratified Richardson number to predict the vertical eddy viscosity ν_e in (6) and the tracer eddy diffusivity κ_e in (10). We first define a "target" eddy diffusivity and eddy viscosity $\kappa_\star$ and $\nu_\star$,

$$\kappa_\star = \kappa_{bg} + \kappa_{\text{conv}} + \kappa_0 \, \text{step}\left(R, R_0, R_\delta\right) \, , \tag{A1}$$

$$\nu_\star = \nu_{bg} + \nu_0 \, \text{step}\left(R, R_0, R_\delta\right) \, , \tag{A2}$$

where $\kappa_{bg} = 10^{-5}\,\mathrm{m^2\,s^{-1}}$ and $\nu_{bg} = 10^{-4}\,\mathrm{m^2\,s^{-1}}$ are constant background mixing coefficients. In (A1)–(A2), step (R, R_0, R_δ) is a smooth step function,

$$\operatorname{step}(R, R_0, R_\delta) \stackrel{\text{def}}{=} \frac{1}{2}\left[1 + \tanh\left(\frac{\langle R\rangle - R_0}{R_\delta}\right)\right], \quad \text{where} \quad R \stackrel{\text{def}}{=} \max\left(0, \frac{N^2}{|\partial_z \boldsymbol{u}_h|^2}\right), \tag{A3}$$

is the Richardson number bounded so that $R \geq 0$ and $N^2 \stackrel{\text{def}}{=} \partial_z b$ is the vertical derivative of buoyancy. The angle brackets $\langle R\rangle$ denote a center-weighted horizontal filter over nine grid points,

$$\begin{aligned}\langle\phi\rangle(x,y) \stackrel{\text{def}}{=}\ & \tfrac{1}{4}\phi(x,y) + \tfrac{1}{8}\phi(x-\Delta x,y) + \tfrac{1}{8}\phi(x+\Delta x,y) \\ & + \tfrac{1}{8}\phi(x,y-\Delta y) + \tfrac{1}{8}\phi(x,y+\Delta y) \\ & + \tfrac{1}{16}\phi(x-\Delta x,y-\Delta y) + \tfrac{1}{16}\phi(x-\Delta x,y+\Delta y) \\ & + \tfrac{1}{16}\phi(x+\Delta x,y-\Delta y) + \tfrac{1}{16}\phi(x+\Delta x,y+\Delta y)\,,\end{aligned} \tag{A4}$$

where Δx and Δy are the horizontal grid spacing in the x and y direction. The horizontal filter helps reduce horizontal noise that appears near the equator . The convective diffusivity κ_{conv} in (A1) is defined via

$$\kappa_{\mathrm{conv}}(z) \stackrel{\text{def}}{=} \begin{cases} \kappa_{ca} & \text{if} \quad N^2(z) < 0 \\ C_{en} J_s^b / N^2 & \text{if} \quad N^2(z) > N_{en}^2 \quad \text{but} \quad N^2(z+\Delta z) < 0\,, \\ 0 & \text{otherwise}\,, \end{cases} \tag{A5}$$

where $\kappa_{ca} = 1.7\,\mathrm{m^2\,s^{-1}}$ is the convective adjustment diffusivity, $N_{en}^2 = 10^{-10}\,\mathrm{s^{-2}}$ is the minimum entrainment layer buoyancy gradient, Δz is the vertical grid spacing, J_s^b is the surface buoyancy flux, and $C_{en} = 0.1$ is the fractional entrainment buoyancy flux compared to the surface buoyancy flux. Finally, $\kappa_\star$ and $\nu_\star$ are averaged in time to obtain the eddy diffusivity and eddy viscosity, such that at each time-step n,

$$\kappa_e^n = \frac{\kappa_\star^n + C_{av}\kappa_\star^{n-1}}{1 + C_{av}}\,, \tag{A6}$$

where $C_{av} = 0.6$. The time-averaging in (A6), which is equivalent to implicitly relaxing the κ_e to the target value $\langle\kappa_\star\rangle$ over a time-scale $C_{av}\,\Delta t$, where Δt is the time-step, helps smooth vertical noise associated with the Richardson-number-based components. The 7 free parameters — C_{av}, C_{en}, κ_0, ν_0, R_0, R_δ, κ_{ca} — are determined by calibration against a set of large eddy simulations, using the same methodology as the one described by Wagner et al. (2024).

Appendix B Additional results from the near-global ocean configuration

In this appendix, we show additional metrics concerning the result of the near-global configuration and its evolution from the initial conditions. These metrics are shown to characterize the time evolution of the model but are not intended to validate the configuration given the known weaknesses of this setup.

Figure B1 shows the time series of integrated global temperature and salinity and integrated global kinetic energy. In the spin-up stage, the model adjusts from the ECCO2 initial conditions towards the new state imposed by the forcing and the parameter choices. The global kinetic energy, shown for the 1/12 degree-configuration, has an initial spin-up phase that lasts around 1.5 years and settles around $37\,\mathrm{cm^2\,s^{-2}}$. Both mean temperature and salinity show a drift with a clear annual cycle. The 1 degree-configuration without mesoscale eddies shows a drastic temperature drift with the global temperature increasing by almost 0.1 °C in 20 years. The global salinity drift is much more contained, with an initial decrease in global salinity subsequently offset by an increase that reduces the global drift. In the 1/12 degree-configuration, the temperature drift is more effectively contained, while

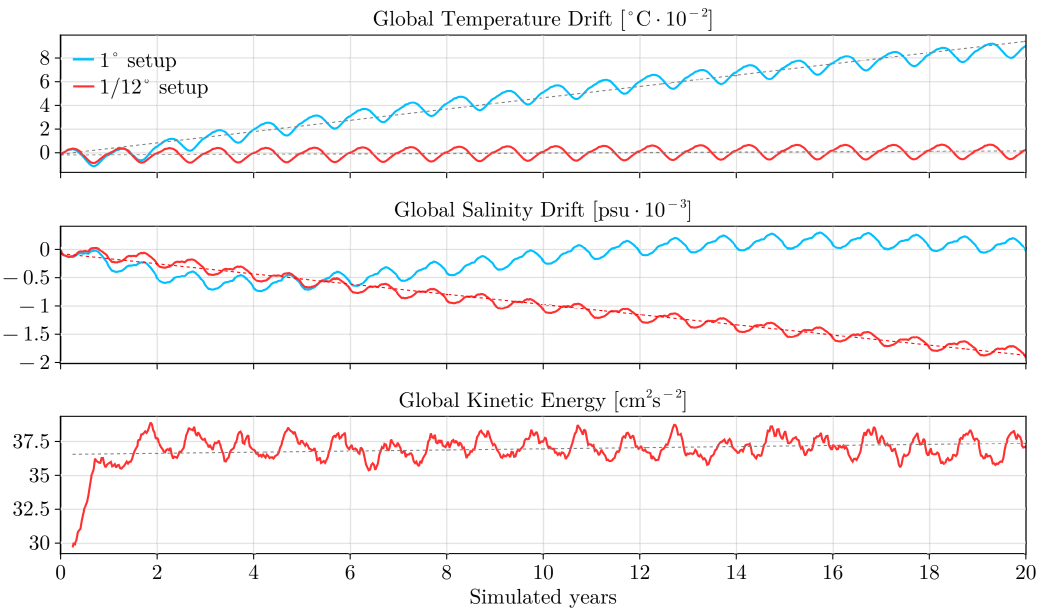

Figure B1: Timeseries of globally averaged temperature (top), salinity (center), and kinetic energy (bottom). The red dashed line shows the best linear fit.

the salinity shows a monotonic decrease with simulation time. After 20 years of evolution, the mean temperature increases by 0.004 °C and the global salinity decreases by about 0.002 psu. These drifts are relatively small and somewhat comparable to those reported by other mesoscale-resolving ocean configurations described in the literature (Iovino et al., 2016).

Figure B2 shows the time series of the Atlantic Meridional Circulation (AMOC) at 26.5° North (top) and the transport across the Drake Passage. The transport across the Drake Passage compares quite well with observations for both the low and the high-resolution configuration. However, the AMOC is mostly determined by the initial conditions evolving with a very slow timescale, much slower than the 20 years of evolution simulated in this setup. Nevertheless, it is crucial to demonstrate that the model preserves the Atlantic circulation. Indeed, the AMOC strength diminishes rapidly in the low-resolution configuration, while it maintains greater intensity in the 1/12 degree-configuration.

This result is confirmed in Figure B3 which presents the structure of the AMOC averaged over the last 10 years of integration. The AMOC is significantly stronger for the eddying solution. The 1/12° model effectively captures the vertical structure of the AMOC, featuring a positive cell extending to approximately four kilometers in depth and maximum transports on the order of 18 Sv. This positive cell is complemented by a lower negative cell with transports ranging between 2 and 4 Sv. The vertical profiles at 26.5° North are compared to the RAPID observations (Johns et al., 2011) on the right of figure B3. The vertical profiles of the AMOC are realistic, although the positive cell's strength is lower than the observed values, with the 1/12° setup being closer to observations.

Open Research Section

Scripts for reproducing the performance tests and the test cases described in this paper are available at Silvestri and Churavy (2024). Visualizations were made using Makie.jl (Danisch & Krumbiegel, 2021).

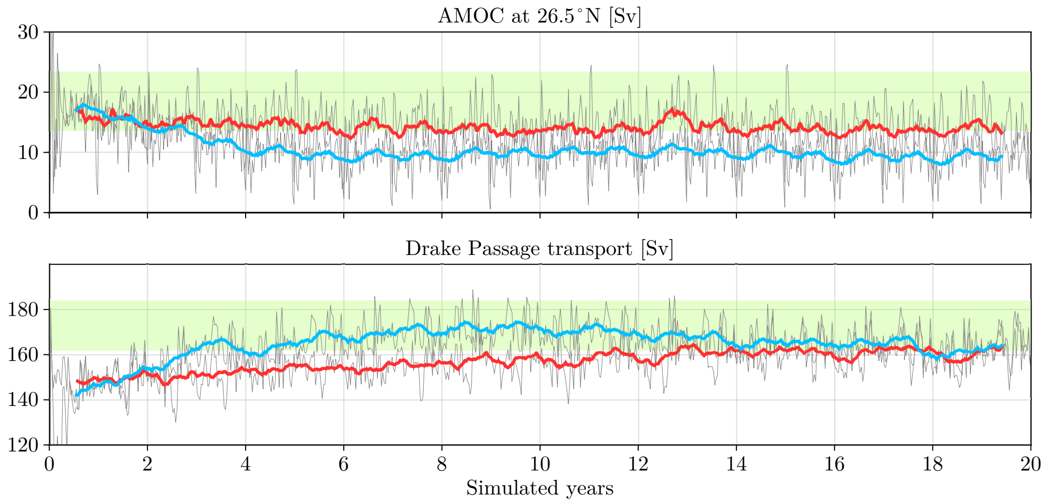

Figure B2: Timeseries of the AMOC strength at 25.6°N (top) and the transport across Drake Passage (bottom). The grey curves show instantaneous 10-day values while the blue and red lines show a 100-day moving average of the 1 degree- and the 1/12 degree-configurations respectively. The shaded areas in the time series show the observed estimates from Johns et al. (2011) (AMOC) and Donohue et al. (2016) (Drake Passage).

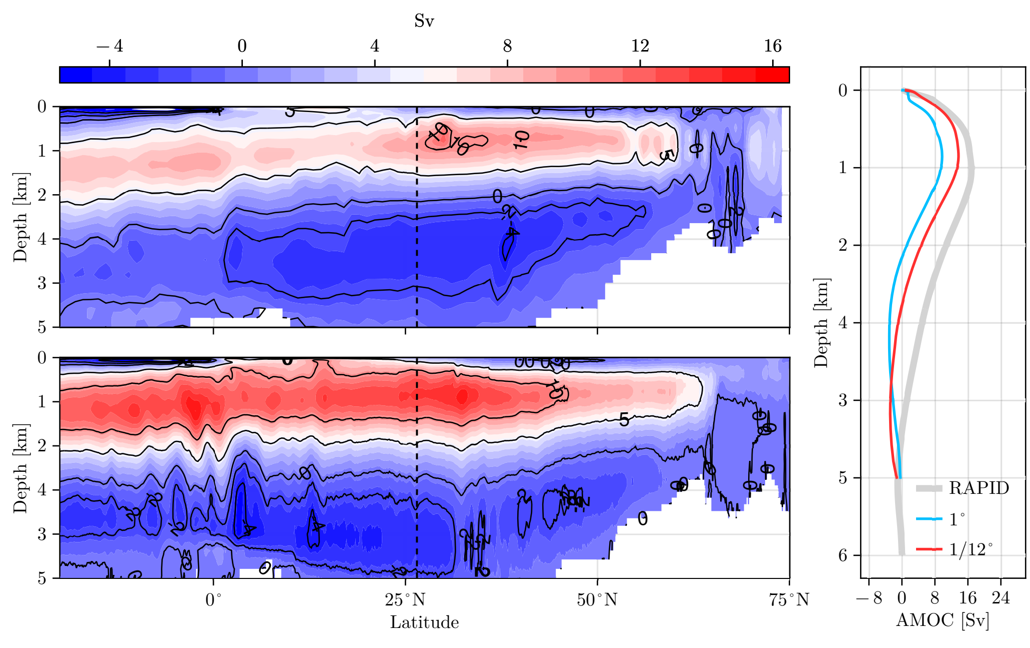

Figure B3: AMOC stream function averaged over the last 10 years of integration for the 1° configuration (top) and the 1/12° configuration (bottom). The plot on the right compares the AMOC vertical structure at 26.5° North with the RAPID-array observations (Johns et al., 2011).

Acknowledgments

This research leveraged the resources of the National Energy Research Scientific Computing Center (NERSC), a premier U.S. Department of Energy Office of Science User Facility at the Lawrence Berkeley National Laboratory, under Contract No. DE-AC02-05CH11231 and NERSC award DDR-ERCAP0025591. Recommended by the Schmidt Futures now Schmidt Sciences program, this work received partial support through the generous contributions of Eric and Wendy Schmidt. We further acknowledge support by the National Science Foundation grant AGS-1835576. N.C.C. was in addition supported by the Australian Research Council under DECRA Fellowship DE210100749 and the Center of Excellence for the Weather of the 21st Century CE230100012. V.C. was supported by the Department of Energy, National Nuclear Security Administration under Award Number DE-NA0003965 and the National Science Foundation under grant No. OAC-2103804.

References

Acosta, M. C., Palomas, S., Ticco, S. V. P., Utrera, G., Biercamp, J., Bretonniere, P.-A., … Balaji, V. (2024). The computational and energy cost of simulation and storage for climate science: lessons from CMIP6. *Geoscientific Model Development*, *17*(8), 3081–3098. doi: 10.5194/gmd-17-3081-2024

Arakawa, A., & Lamb, V. (1977). Computational design of the basic dynamical processes of the UCLA general circulation model. *General Circulation Models of the Atmosphere*, *17*.

AVISO+. (n.d.). *The mesoscale eddy trajectory atlas products were produced by ssalto/duacs and distributed by aviso+ with support from cnes, in collaboration with oregon state university with support from nasa.* Retrieved from https://www.aviso.altimetry.fr

Awan, M., & Saeed, F. (2016). Gpu-arraysort: A parallel, in-place algorithm for sorting large number of arrays. In *2016 45th international conference on parallel processing workshops (icppw)* (p. 78-87). doi: 10.1109/ICPPW.2016.27

Balaji, V., Maisonnave, E., Zadeh, N., Lawrence, B. N., Biercamp, J., Fladrich, U., … others (2017). CPMIP: measurements of real computational performance of Earth system models in CMIP6. *Geoscientific Model Development*, *10*(1), 19–34.

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, *59*(1), 65–98. doi: 10.1137/141000671

Charney, J. G. (1971). Geostrophic turbulence. *Journal of Atmospheric Sciences*, *28*(6), 1087 - 1095. doi: 10.1175/1520-0469(1971)028<1087:GT>2.0.CO;2

Chassignet, E. P., Yeager, S. G., Fox-Kemper, B., Bozec, A., Castruccio, F., Danabasoglu, G., … Xu, X. (2020). Impact of horizontal resolution on global ocean–sea ice model simulations based on the experimental protocols of the Ocean Model Intercomparison Project phase 2 (OMIP-2). *Geoscientific Model Development*, *13*(9), 4595–4637. doi: 10.5194/gmd-13-4595-2020

Christian, T., Lebrun-Grandie, D., Arndt, D., Ciesko, J., Dang, V., Ellingwood, N., … others (2021). Kokkos 3: Programming model extensions for the exascale era. *IEEE Transactions on Parallel and Distributed Systems*, *33*(4), 805–817.

Churavy, V., Aluthge, D., Smirnov, A., Schloss, J., Samaroo, J., Wilcox, L. C., … Haraldsson, P. (2024). *JuliaGPU/KernelAbstractions.jl: v0.9.18.* Zenodo. doi: 10.5281/zenodo.4021259

Constantinou, N. C., & Hogg, A. M. (2021). Intrinsic oceanic decadal variability of upper-ocean heat content. *Journal of Climate*, *34*(15), 6175–6189. doi: 10.1175/JCLI-D-20-0962.1

Costa, P., Phillips, E., Brandt, L., & Fatica, M. (2021). GPU acceleration of CaNS for massively-parallel direct numerical simulations of canonical fluid flows. *Computers & Mathematics with Applications*, *81*, 502-511. doi: https://doi.org/10.1016/j.camwa.2020.01.002

Couespel, D., Lévy, M., & Bopp, L. (2024). Stronger oceanic CO2 sink in eddy-resolving simulations of global warming. *Geophysical Research Letters*, *51*(4), e2023GL106172. doi: 10.1029/2023GL106172

Danisch, S., & Krumbiegel, J. (2021). Makie.jl: Flexible high-performance data visualization for Julia [software]. *Journal of Open Source Software*, *6*(65), 3349. doi: 10.21105/joss.03349

Ding, M., Liu, H., Lin, P., Meng, Y., Zheng, W., An, B., ... others (2022). A century-long eddy-resolving simulation of global oceanic large-and mesoscale state. *Scientific Data*, *9*(1), 691.

Donlon, C., Berruti, B., Mecklenberg, S., Nieke, J., Rebhan, H., Klein, U., ... Seitz, B. (2012). The sentinel-3 mission: Overview and status. In *2012 ieee international geoscience and remote sensing symposium* (pp. 1711–1714).

Donohue, K., Tracey, K., Watts, D., Chidichimo, M., & Chereskin, T. (2016). Mean Antarctic Circumpolar Current transport measured in Drake Passage. *Geophysical Research Letters*, *43*(22), 11760–11767. doi: 10.1002/2016GL070319

Ferrari, R., & Wunsch, C. (2009). Ocean circulation kinetic energy: Reservoirs, sources, and sinks. *Annual Review of Fluid Mechanics*, *41*, 253–282. doi: 10.1146/annurev.fluid.40.111406.102139

Ferreira, D., Marshall, J., & Campin, J.-M. (2010). Localization of deep water formation: Role of atmospheric moisture transport and geometrical constraints on ocean circulation. *J. Climate*, *23*, 1456–1476. doi: 10.1175/2009JCLI3197.1

Fuhrer, O., Chadha, T., Hoefler, T., Kwasniewski, G., Lapillonne, X., Leutwyler, D., ... Vogt, H. (2018). Near-global climate simulation at 1 km resolution: establishing a performance baseline on 4888 GPUs with COSMO 5.0. *Geoscientific Model Development*, *11*(4), 1665–1681. doi: 10.5194/gmd-11-1665-2018

Gadd, A. J. (1978). A split explicit integration scheme for numerical weather prediction. *Quarterly Journal of the Royal Meteorological Society*, *104*(441), 569-582. doi: https://doi.org/10.1002/qj.49710444103

Gent, P. R., & Mcwilliams, J. C. (1990). Isopycnal mixing in ocean circulation models. *Journal of Physical Oceanography*, *20*(1), 150–155. doi: 10.1175/1520-0485(1990)020<0150:IMIOCM>2.0.CO;2

Griffies, S. M., Adcroft, A., & Hallberg, R. W. (2020). A primer on the vertical Lagrangian-remap method in ocean models based on finite volume generalized vertical coordinates. *Journal of Advances in Modeling Earth Systems*, *12*(10), e2019MS001954. doi: 10.1029/2019MS001954

Griffies, S. M., Winton, M., Anderson, W. G., Benson, R., Delworth, T. L., Dufour, C. O., ... Zhang, R. (2015). Impacts on ocean heat from transient mesoscale eddies in a hierarchy of climate models. *Journal of Climate*, *28*(3), 952-977. doi: 10.1175/JCLI-D-14-00353.1

Gupta, M., & contributors. (2024). *HIP: C++ Heterogeneous-Compute Interface for Portability.* https://github.com/ROCm-Developer-Tools/HIP.

Hallberg, R. (2013). Using a resolution function to regulate parameterizations of oceanic mesoscale eddy effects. *Ocean Modelling*, *72*, 92–103.

Hewitt, H. T., Roberts, M., Mathiot, P., Biastoch, A., Blockley, E., Chassignet, E. P., ... Zhang, Q. (2020). Resolving and parameterising the ocean mesoscale in Earth system models. *Current Climate Change Reports*, *6*, 137-152. doi: 10.1007/s40641-020-00164-w

Hu, Y., Huang, X., Wang, X., H. Fu, H., S. Xu, S., Ruan, H., ... Yang, G. (2013). A scalable barotropic mode solver for the parallel ocean program. In *Euro-par 2013 parallel processing* (pp. 739–750). Berlin, Heidelberg: Springer Berlin Heidelberg.

Häfner, D., Nuterman, R., & Jochum, M. (2021). Fast, cheap, and turbulent—global ocean modeling with GPU acceleration in Python. *Journal of Advances in Modeling Earth Systems*, *13*(12), e2021MS002717. doi: 10.1029/2021MS002717

Ingleby, B., & Huddleston, M. (2007). Quality control of ocean temperature and salinity profiles — historical and real-time data. *Journal of Marine Systems*, *65*(1), 158-175. (Marine Environmental Monitoring and Prediction) doi: 10.1016/j.jmarsys.2005.11.019

Iovino, D., Masina, S., Storto, A., Cipollone, A., & Stepanov, V. (2016). A 1/16 eddying simulation of the global nemo sea-ice–ocean system. *Geoscientific Model Development*, *9*(8), 2665–2684. doi: 10.5194/gmd-9-2665-2016

Jakob, W. (2019). *Enoki: structured vectorization and differentiation on modern processor architectures.* (https://github.com/mitsuba-renderer/enoki)

Johns, W., Baringer, M., Beal, L., Cunningham, S., Kanzow, T., Bryden, H., . . . Curry, R. (2011). Continuous, array-based estimates of Atlantic Ocean heat transport at 26.58N. *Journal of Climate*, 2429–2449. doi: 10.1175/2010JCLI3997.1

Kang, H.-G., Evans, K., Petersen, M., Jones, P., & Bishnu, S. (2021). A scalable semi-implicit barotropic mode solver for the mpas-ocean. *Journal of Advances in Modeling Earth Systems*, *13*(4), e2020MS002238. doi: https://doi.org/10.1029/2020MS002238

Kay, J. E., Deser, C., Phillips, A., Mai, A., Hannay, C., Strand, G., . . . Vertenstein, M. (2015). The Community Earth System Model (CESM) Large Ensemble Project: A Community Resource for Studying Climate Change in the Presence of Internal Climate Variability. *Bulletin of the American Meteorological Society*, *96*(8), 1333 - 1349. doi: 10.1175/BAMS-D-13-00255.1

Killworth, P. D., Webb, D. J., Stainforth, D., & Paterson, S. M. (1991). The development of a free-surface Bryan–Cox–Semtner ocean model. *Journal of Physical Oceanography*, *21*(9), 1333–1348.

Kiss, A. E., Hogg, A. M., Hannah, N., Boeira Dias, F., Brassington, G. B., Chamberlain, M. A., . . . Zhang, X. (2020). ACCESS-OM2 v1.0: a global ocean–sea ice model at three resolutions. *Geoscientific Model Development*, *13*(2), 401–442. doi: 10.5194/gmd-13-401-2020

Kochkov, D., Yuval, J., Langmore, I., Norgaard, P., Smith, J., Mooers, G., . . . Hoyer, S. (2024). Neural general circulation models for weather and climate. *Nature*, *632*, 1060–1066. doi: 10.1038/s41586-024-07744-y

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90.

Marshall, J., Adcroft, A., Hill, C., Perelman, L., & Heisey, C. (1997). A finite-volume, incompressible Navier Stokes model for studies of the ocean on parallel computers. *Journal of Geophysical Research*, *102*, 5753-5766. doi: 10.1029/96JC02775

McClean, J. L., Bader, D. C., Bryan, F. O., Maltrud, M. E., Dennis, J. M., Mirin, A. A., . . . Worley, P. H. (2011). A prototype two-decade fully-coupled fine-resolution CCSM simulation. *Ocean Modelling*, *39*(1), 10-30. doi: 10.1016/j.ocemod.2011.02.011

Menemenlis, D., Campin, J.-M., Heimbach, P., Hill, C., Lee, T., Nguyen, A., . . . Zhang, H. (2008). Ecco2: High resolution global ocean and sea ice data synthesis. *Mercator Ocean Quarterly Newsletter*, *31*(October), 13–21.

Micikevicius, P. (2010). Analysis-driven optimization driven optimization (GTC 2010). In *Nvidia Graphics Technology Conference (gtc).* Retrieved from https://www.nvidia.com/content/gtc-2010/pdfs/2012_gtc2010.pdf

Morrow, R., Fu, L.-L., Ardhuin, F., Benkiran, M., Chapron, B., Cosme, E., . . . Zaron, E. (2019). Global observations of fine-scale ocean surface topography with the Surface Water and Ocean Topography (SWOT) mission. doi: 10.3389/fmars.2019.00232

NVIDIA Corporation. (2010). *NVIDIA CUDA C programming guide.* (Version 3.2)

Palmer, T. (2014). Build high-resolution global climate models. *Nature*, *515*, 338–339.

Prims, O. T., Acosta, M., Moore, A., Castrillo, M., Serradell, K., Cortés, A., & Doblas-Reyes, F. (2019). How to use mixed precision in ocean models: exploring a potential reduction of numerical precision in nemo 4.0 and roms 3.6. *Geoscientific Model Development*, *12*(7), 3135–3148. Retrieved from https://gmd.copernicus.org/articles/12/3135/2019/ doi: 10.5194/gmd-12-3135-2019

Rackauckas, C. (2023). Jax vs PyTorch vs Julia GPU Benchmarks (Peer Reviewed), ODE Solvers, AI for Science and SciML. doi: 10.6084/m9.figshare.24586980.v1

Raina, R., Madhavan, A., & Ng, A. Y. (2009). Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on* *machine learning* (pp. 873–880).

Ramadhan, A., Wagner, G., Hill, C., Campin, J.-M., Churavy, V., Besard, T., … Marshall, J. (2020). Oceananigans.jl: Fast and friendly geophysical fluid dynamics on GPUs. *Journal of Open Source Software*, *5*(53).

Räss, L., Omlin, S., & Podladchivok, Y. (2019). Porting a massively parallel multi-gpu application to julia: a 3-d nonlinear multi-physics flow solver. In *Juliacon conference.* Baltimore, USA.

Ringler, T., Petersen, M., Higdon, R. L., Jacobsen, D., Jones, P. W., & Maltrud, M. (2013). A multi-resolution approach to global ocean modeling. *Ocean Modelling*, *69*, 211-232. doi: 10.1016/j.ocemod.2013.04.010

Roberts, M. J., Vidale, P. L., Senior, C., Hewitt, H. T., Bates, C., Berthou, S., … Wehner, M. F. (2018). The benefits of global high resolution for climate simulation: Process understanding and the enabling of stakeholder decisions at the regional scale. *Bulletin of the American Meteorological Society*, *99*(11), 2341-2359. doi: 10.1175/BAMS-D-15-00320.1

Roquet, F., Madec, G., McDougall, T., & Barker, P. (2015, 04). Accurate polynomial expressions for the density and specific volume of seawater using the teos-10 standard. *Ocean Modelling*. doi: 10.1016/j.ocemod.2015.04.002

Sætra, M. L. (2013). Shallow water simulation on gpus for sparse domains. In A. Cangiani, R. Davidchack, E. Georgoulis, A. Gorban, J. Levesley, & M. Tretyakov (Eds.), *Numerical mathematics and advanced applications 2011* (pp. 673–680). Berlin, Heidelberg: Springer Berlin Heidelberg.

Sakharnykh, N. (2009). Tridiagonal solvers on the GPU and applications to fluid simulation. In *GPU Technology Conference.* Retrieved from https://www.nvidia.com/content/gtc/documents/1058_gtc09.pdf

Schaller, R. R. (1997). Moore's law: past, present and future. *IEEE spectrum*, *34*(6), 52–59.

Schneider, T., Lan, S., Stuart, A., & Teixeira, J. (2017). Earth System Modeling 2.0: A blueprint for models that learn from observations and targeted high-resolution simulations. *Geophysical Research Letters*, *44*(24), 12–396.

Shchepetkin, A. F., & McWilliams, J. C. (2005). The regional oceanic modeling system (ROMS): a split-explicit, free-surface, topography-following-coordinate oceanic model. *Ocean modelling*, *9*(4), 347–404. doi: 10.1016/j.ocemod.2004.08.002

Silvestri, S., & Churavy, V. (2024, September). *Oceanscalingtests.jl.* Zenodo. Retrieved from https://doi.org/10.5281/zenodo.13839329 doi: 10.5281/zenodo.13839329

Silvestri, S., & Pecnik, R. (2019). A fast gpu monte carlo radiative heat transfer implementation for coupling with direct numerical simulation. *Journal of Computational Physics: X*, *3*, 100032. doi: https://doi.org/10.1016/j.jcpx.2019.100032

Silvestri, S., Wagner, G., Campin, J.-M., Constantinou, N., Hill, C., Souza, A., & Ferrari, R. (2024). A new WENO-based momentum advection scheme for simulations of ocean mesoscale turbulence. *Journal of Advances in Modeling Earth Systems*, *16*(7), e2023MS004130. doi: 10.1029/2023MS004130

Singh, P., Sukumaran-Rajam, A., Rountev, A., Rastello, F., Pouchet, L.-N., & Sadayappan, P. (2018, February). Register Optimizations for Stencils on GPUs. In *PPoPP 2018 - 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (p. 1-15). Vienna, Austria.

Souza, A., He, J., Bischoff, T., Waruszewski, M., Novak, L., Barra, V., … Schneider, T. (2023). The flux-differencing discontinuous galerkin method applied to an idealized fully compressible nonhydrostatic dry atmosphere. *Journal of Advances in Modeling Earth Systems*, *15*(4), e2022MS003527. doi: https://doi.org/10.1029/2022MS003527

Sridhar, A., Tissaoui, Y., Marras, S., Shen, Z., Kawczynski, C., Byrne, S., … Schneider, T. (2022). Large-eddy simulations with climatemachine v0.2.0: a new open-source code for atmospheric simulations on gpus and cpus. *Geoscientific Model Development*, *15*(15), 6259–6284. Retrieved from https://gmd.copernicus.org/articles/15/6259/2022/ doi: 10.5194/gmd-15-6259-2022

Sutter, H., et al. (2005). The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's journal*, *30*(3), 202–210.

Taylor, J., & Thompson, A. (2023). Submesoscale Dynamics in the Upper Ocean. *Annual Review of Fluid Mechanics*, *55*(1), 103-127. doi: 10.1146/annurev-fluid-031422-095147

Taylor, M., Caldwell, P., Bertagna, L., Clevenger, C., Donahue, A., Foucar, J., ... Wu, D. (2023). The simple cloud-resolving E3SM atmosphere model running on the Frontier exascale system. In *Proceedings of the international conference for high performance computing, networking, storage and analysis.* New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3581784.3627044

Tran, N.-P., Lee, M., & Hong, S. (2017). Performance optimization of 3d lattice boltzmann flow solver on a gpu. *Scientific Programming*, *2017*(1), 1205892. doi: https://doi.org/10.1155/2017/1205892

Vance, A. (2009). *Hello, dally: Nvidia scientist breaks silence, criticizes intel.* Retrieved from https://archive.nytimes.com/bits.blogs.nytimes.com/2009/04/09/hello-dally-nvidia-scientist-breaks-silence-criticizes-intel/

Wagner, G. L., Hillier, A., Constantinou, N. C., Silvestri, S., Souza, A., Burns, K., ... Ferrari, R. (2024). Formulation and calibration of CATKE, a one-equation parameterization for microscale ocean mixing. *arXiv*. (submitted to J. Adv. Model. Earth Sy.) doi: 10.48550/arXiv.2306.13204

Wang, G., Lin, Y., & Yi, W. (2010). Kernel fusion: An effective method for better power efficiency on multithreaded gpu. In *2010 ieee/acm international conference on green computing and communications* (p. 344-350). doi: 10.1109/GreenCom-CPSCom.2010.102

Wang, P., Jiang, J., Lin, P., Ding, M., Wei, J., Zhang, F., ... Liu, H. (2021). The GPU version of LASG/IAP climate system ocean model version 3 (LICOM3) under the heterogeneous-compute interface for portability (HIP) framework and its large-scale application. *Geosci. Model Dev.*, *14*(5), 2781–2799. doi: 10.5194/gmd-14-2781-2021

Wei, J., Jiang, J., Liu, H., Zhang, F., Lin, P., Wang, P., ... Wang, Y. (2023). LICOM3-CUDA: a GPU version of LASG/IAP climate system ocean model version 3 based on CUDA. *The Journal of Supercomputing*, *79*(9), 9604–9634. doi: 10.1007/s11227-022-05020-2