Master in High Performance Computing

Extending Oceananigans.jl Ocean Model Toward Regional Applications: Evaluation and Implementation of Radiative Open Boundary Schemes

Supervisor(s):
Celia Laurent,
Simone Silvestri,
Irina Davydenkova

Candidate:
Nicola Aladrah

11th edition
2024–2025

Contents

Abstract

General circulation models (GCMs) are widely used in oceanography to study climate processes and regional hydrodynamics. Recent advances in high-performance computing and modern programming languages provide new opportunities to develop flexible and efficient modeling tools. Oceananigans.jl, written in Julia, is a high-performance ocean model designed to run efficiently on both CPUs and GPUs while allowing modular and customizable configurations. This thesis contributes to extending Oceananigans.jl for regional ocean simulations by implementing radiative open boundary conditions. First, a one-dimensional advection-based scheme inspired by Orlanski-type radiation is developed and validated in an idealized configuration. The method is then extended to two dimensions using a Raymond–Kuo–type formulation and tested in a North Adriatic Sea setup forced by Copernicus Marine Service hydrodynamic fields along the open boundary.

Finally, the execution behavior of the enhanced solver is examined on GPU architectures, in particular NVIDIA A100 GPUs on the Leonardo supercomputer at CINECA, demonstrating the practical portability of `Oceananigans.jl` to modern GPU-based high-performance computing systems for regional ocean modeling.

Thesis Organization

This thesis is organized as follows.

Chapter 1 introduces the scientific and computational context of regional ocean modeling and motivates the need for open boundary conditions in nested regional configurations. It outlines the challenges associated with wave radiation, inflow control, and numerical stability, and states the concrete objectives of the thesis.

Chapter 2 reviews the theoretical background of open boundary conditions. It introduces the Sommerfeld radiation condition, discusses its interpretation in one spatial dimension, and summarizes classical extensions to multidimensional settings. This chapter establishes the mathematical form of the boundary condition that guides the algorithmic developments in later chapters.

Chapter 3 presents the development of an advection-based open boundary condition tailored to the hydrostatic free-surface solver in `Oceananigans.jl`. The one-dimensional formulation is derived first, including the phase-speed diagnostic, CFL bounding, and implicit boundary update. The method is then extended to two dimensions using a Raymond–Kuo style formulation that incorporates tangential gradients and optional nudging. Algorithmic details and implementation choices are discussed alongside the mathematical formulation.

Chapter 4 focuses on numerical experiments and validation. Idealized one-dimensional barotropic tests are used to assess the effectiveness of the boundary condition in radiating outgoing disturbances. The chapter then presents two-dimensional experiments in a North Adriatic Sea configuration, forced by external oceanographic and atmospheric data. The observed stability limits and failure modes are analyzed in detail.

Chapter 5 addresses performance and benchmarking. It describes the computational setup on the Leonardo (CINECA) system, explains how the same Julia code is executed on CPU and GPU architectures, and presents benchmark results using simulated years per wall day as the performance metric. The chapter demonstrates substantial GPU acceleration and discusses the implications for regional ocean modeling workflows.

Chapter 6 concludes the thesis. It summarizes the main scientific and computational findings, discusses current limitations of the two-dimensional boundary implementation, and outlines directions for future work, including algorithmic extensions, solver improvements, and large-scale GPU and MPI benchmarking.

Chapter 1

Regional Ocean Modeling: Context and challenges

Regional ocean models are widely used to study coastal circulation, climate variability, and marine ecosystems. These models focus computational resources on a limited geographical domain in order to resolve mesoscale as well as large scale and coastal processes that cannot be captured at global resolution.

Most existing ocean circulation models were originally developed as large monolithic Fortran codes optimized for earlier CPU architectures. While these models remain widely used, their structure can make the integration of new algorithms and adaptation to modern heterogeneous computing architectures more difficult.

`Oceananigans.jl` is a recent ocean modeling framework written in Julia and designed for performance portability across CPUs and GPUs. Its modular library-style architecture allows flexible implementation of numerical schemes and physical parameterizations. However, support for realistic regional configurations is still evolving, particularly regarding radiative open boundary conditions required to connect regional simulations with larger-scale ocean models.

Developing robust open boundary conditions for Oceananigans is therefore an important step toward enabling realistic regional simulations within this modern computational framework.

1.1 The Computational Imperative in Regional Oceanography

The numerical simulation of ocean circulation stands as a cornerstone of modern geophysical fluid dynamics, providing the essential predictive capacity required for understanding climate variability, managing marine ecosystems, and mitigating coastal hazards. While Global Circulation Models (GCMs) have achieved remarkable success in resolving the large-scale state of the ocean, they inevitably face a resolution barrier. The computational cost of resolving small-scale processes–such as submesoscales, coastal trapped waves, estuarine fronts, and topographic eddies–on a global grid remains prohibitively high for multi-decadal studies[1]. Consequently, the oceanographic community relies heavily on regional ocean modeling, a downscaling technique where high-resolution computational resources are focused on a specific domain of interest, such as the Adriatic Sea or the Mediterranean Sea, while the influence of the global ocean is parameterized through lateral boundaries[2].

This truncation of the physical domain introduces one of the most persistent and theoretically complex challenges in computational fluid dynamics: the specification of Open Boundary Conditions (OBCs). Unlike closed basins bounded by rigid coastlines where no-slip or free-slip conditions provide mathematically well-posed constraints, an open boundary is a purely artificial interface[3] where the model must be kept close to observations or reference fields provided by larger scale models. It must function as a semi-permeable membrane, allowing phenomena generated within the high-resolution domain—such as gravity waves, geostrophic turbulence, and advected tracers—to exit the grid without spurious reflection, while simultaneously permitting information from the external environment (e.g., tides, surges, large-scale circulation patterns, temperature, salinity, and other relevant flow characteristics and tracers) to enter and force the interior dynamics[4].

The failure to properly treat these boundaries results in severe numerical artifacts. Artificial reflections can trap wave energy inside the limited area, leading to resonance, noise accumulation, and eventually catastrophic model instability. Thus, the development of robust, stable, and accurate OBCs is not merely a technical detail but a fundamental prerequisite for the validity of any regional ocean simulation.

1.2 `Oceananigans.jl` and Modern Architectures

Since decades, the landscape of regional ocean modeling is dominated by state-of-the-art, long-standing and extensively validated Fortran-based codes such as the Regional Ocean Modeling System (ROMS)[5], NEMO – Nucleus for European Modelling of the Ocean [6] and the MIT General Circulation Model (MITgcm) [7]. These models represent the culmination of years of community development and feature extensive options of parameterizations and boundary conditions. However, they are also constrained by the legacy of their architectural origins, often facing challenges in adapting to modern heterogeneous computing environments, specifically the massive parallelism offered by Graphics Processing Units (GPUs)[1].

The emergence of Oceananigans.jl represents a paradigm shift in this domain. Developed in the Julia programming language, Oceananigans.jl is designed from the ground up for performance portability and high-level abstraction. It allows researchers to write expressive, high-level code that runs efficiently on both CPUs and GPUs without significant modification. This architecture offers profound advantages, particularly in the non-hydrostatic regime where vertically accelerated motions and fine-scale dynamics can be explicitly represented at sufficiently high resolution. By leveraging the abstract calculus of Julia and the inherent parallelism of GPUs, Oceananigans.jl enables simulations of unprecedented resolution and speed, improving simulation quality by better resolving small-scale dynamics, particularly in coastal regions where such processes are crucial.

However, as a relatively nascent platform compared to ROMS, NEMO or MITgcm, Oceananigans.jl is still expanding its capability for realistic regional configurations. While it excels in idealized process studies (e.g., large eddy simulation of turbulence), its infrastructure for handling complex, data-forced open boundaries in realistic regional

domains is an active area of development. The implementation of sophisticated OBC schemes, such as the Orlanski radiation condition, is critical to transitioning Oceananigans.jl from a model designed primarily for global simulations to a viable engine for operational regional oceanography.

1.3 Case Study: The North Adriatic Sea

To validate the new implementation of open boundary conditions within this modern framework, it was necessary to consider a test-case basin exhibiting both complex, characteristic circulation and a strong sensitivity to open-boundary dynamics. The North Adriatic Sea was selected for this purpose. This shallow, semi-enclosed basin is strongly connected to the southern Adriatic basin through a large open boundary with waters entering the domain by flowing northward along the Croatian coast and exiting flowing southward along the Italian Shelf, following the global anti-clockwise circulation of the Adriatic Sea. The North Adriatic Sea represents a challenging testbed for numerical models due to its complex hydrodynamics, which are driven by a dynamic interplay of strong atmospheric forcing, significant freshwater discharge, and a strong influence of the broader Adriatic circulation.

In winter, northeasterly winds (Bora) and atmospheric forcing induce strong cooling and evaporation over the northern Adriatic basin, increasing surface water density. These dense waters sink to the bottom, forming what is called the Northern Adriatic Dense Water (NADW), flowing southward along the Adriatic slope, following the Italian continental margin [8, 9].

The vertical structure at the open boundary, with a deep southward dense current beneath a more variable upper-ocean circulation, together with the surface anticyclonic circulation described earlier, impose stringent requirements on the boundary condition, which must be capable of both radiate outflows and accommodate for inner fluxes forced by the general circulation and the vertical density gradients.

Chapter 2

Literature Review: Architectures

Open boundaries are a basic problem of ill-posedness in hyperbolic and parabolic systems, not just a discretization problem. The road map for stabilizing contemporary high-performance codes is provided by the techniques created over the previous forty years to address this, ranging from straightforward radiation conditions to intricate relaxation schemes. The treatment of open boundaries in ocean modeling is intrinsically related to the mathematical well-posedness of the underlying equations, and cannot be regarded purely as a numerical discretization problem. The improper specification of boundary conditions may result in ill-posed or unstable solutions. To address these issues, a variety of techniques have been developed over the past decades, ranging from radiation conditions to more elaborate relaxation schemes, and are now commonly employed in contemporary models.

2.1 Orlanski boundary condition based on Sommerfeld radiation condition

The Sommerfeld radiation condition serves as the theoretical foundation for the great majority of "passive" open boundary conditions used in oceanography. It offers a mathematical expression of the physical intuition that all energy must propagate away from the source at infinity. It was first developed in the context of electromagnetic scattering. This condition is used at the artificial boundary in a finite difference or finite volume ocean model to guarantee that outgoing waves pass through transparently.

Following Orlanski [4], the Sommerfeld radiation condition can be expressed in the simple one-dimensional form

$$\frac{\partial \phi}{\partial t} + c_\phi \frac{\partial \phi}{\partial e} = 0 \tag{2.1}$$

Where:

- ϕ represents the prognostic variable (e.g., sea surface elevation η, velocity components u, v, or tracers T, S).
- e denotes the direction normal to the boundary (positive outward).
- c_ϕ is the phase velocity of the disturbance at the boundary.

According to this equation, the propagation of the wave shape out of the domain at speed c_ϕ determines the time rate of change of the variable ϕ at the boundary. The boundary becomes perfectly transparent if c_ϕ is appropriately specified, matching the actual phase speed of the internal waves approaching the boundary.

The establishment that the phase speed c_ϕ in a stratified, rotating fluid is not a constant known a priori was the crucial breakthrough made by Orlanski [4]. The particular mode of propagation—such as advective currents, slow baroclinic internal waves, or fast barotropic gravity waves—determines this. Orlanski suggested a numerical technique to use the interior solution history of the model to dynamically diagnose this phase speed.

The implementation of the Orlanski radiation scheme in practical ocean models is typically carried out in a predictor–corrector fashion:

1. By flipping the Sommerfeld equation around, the phase velocity can be defined using the grid points just inside the boundary—specifically at $n-1$ and $n-2$—across the last couple of time steps. The diagnostic phase speed, c_{diag}, states how fast a disturbance is heading toward the exit:

$$c_{diag} = -\frac{\partial\phi/\partial t}{\partial\phi/\partial e} \tag{2.2}$$

2. In order to prevent the simulation from crashing due to CFL instability, the speed has to be non-negative—since it considers only waves actually leaving the domain—and it cannot exceed the maximum speed the grid can handle. Specifically, it must stay within the bounds of the CFL condition to ensure stability:

$$0 \leq c_{diag} \leq \frac{\Delta x}{\Delta t} \tag{2.3}$$

3. The bounded phase speed c_{diag} is then substituted back into equation 2.1 in place of the phase velocity c_ϕ to update the boundary value of the field at the next time step $t+1$.

2.2 The Raymond-Kuo Generalization

The classical one–dimensional radiation condition derived from the Sommerfeld outgoing–wave relation introduced equation 2.1 assumes that propagation occurs purely in the normal direction following a one–dimensional radiation condition. Using a simplified notation for a diagnostic variable ϕ, it can be expressed as:

$$\partial_t\phi + c_{\text{diag}}\partial_e\phi = 0. \tag{2.4}$$

This form assumes that disturbances propagate purely in the normal direction, however, Raymond and Kuo [10] pointed out that in multidimensional flows disturbances generally approach the boundary with oblique incidence.

The authors indicate that the propagation direction contains both normal and tangential components, so that purely one–dimensional radiation conditions are insufficient to represent the true propagation of the signal. To address this limitation they introduced a multidimensional generalization of the radiation boundary condition.

Considering that t is time, e is the coordinate along the outward normal direction to the boundary, and the two tangential coordinates τ_1 and τ_2, where $\phi = \phi(t, \tau_1, \tau_2, e)$, Raymond and Kuo [10] express the multidimensional radiation relation as

$$\partial_t \phi + c_{\tau_1} \partial_{\tau_1} \phi + c_{\tau_2} \partial_{\tau_2} \phi + c_e \partial_e \phi = 0. \tag{2.5}$$

and proposed a formulation for diagnosing the velocity components directly from the resolved solution.

Based on the work of Raymond and Kuo, Marchesiello et al. [11] formalize and implement the following two–dimensional formulation considering only the horizontal gradients, which is typically sufficient in practical ocean applications for diagnosing propagation along the open boundaries:

$$\partial_t \phi + c_x \partial_x \phi + c_e \partial_e \phi = 0. \tag{2.6}$$

This equation expresses the assumption that the local time tendency of the field can be explained by advection of the signal with a phase velocity vector (c_x, c_e). The coefficients c_x and c_e therefore represent the components of the propagation velocity of the disturbance along the tangential and normal directions, respectively.

Using the closure assumption that propagation velocity is assumed to be aligned with the local gradient of the field, the multidimensional radiation equation provides one scalar relation for the unknown velocity components c_x and c_e.

$$c_x = -\frac{\partial_t \phi \, \partial_x \phi}{(\partial_x \phi)^2 + (\partial_e \phi)^2}, \qquad c_e = -\frac{\partial_t \phi \, \partial_e \phi}{(\partial_x \phi)^2 + (\partial_e \phi)^2}. \tag{2.7}$$

These expressions adopted in the open boundary condition scheme of Marchesiello et al. [11] can provide a unified boundary framework adequate for all prognostic variables ϕ of the ocean model, including baroclinic and barotropic horizontal velocity components, free surface elevation, and tracers such as temperature and salinity.

2.3 The Regional Ocean Modeling System (ROMS) Approach

ROMS is one of the standard-bearer for regional ocean modeling, widely used for coastal applications similar to the North Adriatic Sea. Its handling of open boundaries has evolved significantly from pure radiation to complex hybrid schemes, offering a blueprint to provide a stable open-boundary scheme using nudging and suited for the hydrostatic solver in Oceananigans.

2.3.1 The Radiation-Nudging (RadNud) Paradigm

As described by Marchesiello et al. [11], while radiation conditions are necessary for handling outgoing transients, they are insufficient for long-term stability of the numerical models and do not constrain the solution toward an external state. For regional models, pure radiation allows the interior solution to drift indefinitely, unconstrained by the external reality provided by large-scale models. Furthermore, radiation conditions are physically invalid during inflow events.

Following Marchesiello et al. [11] ROMS does not restrict radiation–nudging to tracers alone. Rather, the barotropic variables are treated separately, with Chapman and Flather conditions for free surface and depth-integrated normal velocity, while the baroclinic prognostic fields are handled through radiation–nudging open-boundary conditions, where the radiation diagnosis may include both normal and tangential gradients.

In the ROMS implementation, the switch between weak outflow nudging and strong inflow nudging is not based on a separate comparison between phase-speed direction and local normal velocity. Instead, it is diagnosed from the sign of the product between the local temporal tendency and the interior normal gradient. Denoting these by $\partial_t\phi$ and $\partial_n\phi$, respectively, the code applies the inflow timescale when

$$(\partial_t\phi)(\partial_n\phi) < 0,$$

and the outflow timescale otherwise. In the incoming case, the radiative tendency is additionally suppressed in the update, so that the boundary is controlled primarily by relaxation toward the external data. Therefore, the mixed cases posed in terms of "outward phase speed but inward local velocity", or conversely, are not treated as separate branches in this implementation. They are absorbed into a single characteristic-based diagnostic. Since this criterion depends on discrete local derivatives, its sign may fluctuate when the solution is weak or noisy, which can in principle lead to switching between inflow- and outflow-type behavior from one time step to the next.

2.3.2 Barotropic OBCs

ROMS uses Barotropic vs. Baroclinic mode-splitting technique. It separates the fast, depth-integrated (barotropic) mode from the slow, depth-dependent (baroclinic) mode, assuming that non-dispersive (i.e. shallow water) surface gravity waves account for a large part of the transient barotropic signals calculated in a regional configuration. The boundary conditions for these modes are treated differently to ensure stability. While the baroclinic mode employ the RadNud paradigm, the barotropic open boundary conditions rely on the Chapman - Flather formulations:

- **Chapman Condition (η):** For the free surface elevation, ROMS uses the Chapman condition [12], which assumes that all perturbations leave the domain as shallow water waves at speed $C = \sqrt{gH}$. This is more robust than diagnosing the speed via Orlanski, as the gravity wave speed is physically fixed by depth. This replaces diagnosing a local phase speed (as in Orlanski schemes) with a physically prescribed wave speed derived from the shallow water equations. In differential form, the Chapman condition can be written for η as a shallow water wave relation:

$$\frac{\partial \eta}{\partial t} = \pm\sqrt{gH}\frac{\partial \eta}{\partial e} \tag{2.8}$$

- **Flather Condition ($\bar{u}$):** For barotropic momentum, the Flather condition [13] is widely regarded as the most stable. The Flather boundary condition applies to the barotropic velocity component normal to the boundary u_n, and it is designed to let barotropic momentum anomalies (including tidal currents) exit (radiation) while anchoring inflow components to external information (dirichlet). In contrast to Chapman, which operates on the free surface, Flather directly couples the surface

elevation difference between the model and the external forcing to the normal barotropic velocity:

$$\bar{u}_n = \bar{u}_n^{ext} - \sqrt{\frac{g}{H}}(\eta - \eta^{ext}) \tag{2.9}$$

It enforces the radiation condition on velocity but adds a correction term based on the difference between the model sea level and the external sea level. This ensures that the mass transport across the boundary is consistent with the external forcing, preventing the "empty ocean" problem where mass slowly leaks out of the domain.

2.4 Oceananigans vs. Legacy Architectures

The transition from a 1D–to–2D regional model requires navigating the significant architectural differences between `Oceananigans.jl` and the legacy codes such as `ROMS` or `MITgcm` that have historically defined the field. While Oceananigans offers superior performance on modern hardware, it currently lacks the extensive library of pre-configured "regional oceanography" modules found in its predecessors.

2.4.1 High-Level Abstraction on GPUs

`Oceananigans.jl` distinguishes itself from ROMS and MITgcm through its design for performance portability. While legacy codes are typically written in Fortran with MPI parallelism rooted in the architectures of the 1990s, Oceananigans is written in Julia, leveraging the language's abstract calculus to generate optimized kernels for both CPUs and GPUs. This allows for a *"write once, run anywhere"* workflow where the same boundary condition code can execute on a laptop CPU for debugging and a cluster GPU for production.

For regional modeling, Oceananigans employs the `HydrostaticFreeSurfaceModel`. This solver is mathematically analogous to the split-explicit kernels of ROMS, solving the hydrostatic Boussinesq equations with a free surface. However, unlike ROMS, which provides a monolithic executable controlled by text input files, Oceananigans functions as a library, offering immense flexibility for additional schemes implementation, applications and parameterization.

2.4.2 Boundary Condition as Primitives vs. Modules

The most critical divergence lies in how boundary conditions are exposed to the user.

ROMS/MITgcm (The Modular Approach). In ROMS, open boundaries are selected via high-level keywords in a configuration file (e.g., `.in` file). To implement a stable regional boundary, a user might simply select:

- `LBC(isUbar)== Flather` (for barotropic velocity)
- `LBC(isTvar)== RadNud` (for tracers)

Where Fortran code automatically handles the complex logic of diagnosing phase speeds, switching between inflow and outflow nudging timescales, and applying volume

conservation corrections. The "RadNud" scheme, which is the gold standard for stability, is a pre-packaged feature.

`Oceananigans.jl` **(The Primitive Approach).** Oceananigans does not yet possess these keywords for regional modeling. Instead, it provides low-level boundary condition *primitives*:

- `FluxBoundaryCondition`: Prescribes a flux (Neumann).
- `ValueBoundaryCondition`: Prescribes a value (Dirichlet).
- `GradientBoundaryCondition`: Prescribes a gradient.
- `OpenBoundaryCondition`: A special type allowing the user to modify halo regions directly.

To replicate the functionality of ROMS's `RadNud` in Oceananigans, a function that performs the diagnosis of the phase speed must be implemented, checking the direction of flow, and conditionally applying either radiation or nudging. This gap between the raw boundary primitives and a fully featured regional boundary scheme is what the developer must implement to achieve adequate boundary behavior while maintaining stability.

Chapter 3

Numerical Implementation and Algorithmic Improvments

While `Oceananigans.jl` provides a robust kernel for non-hydrostatic and hydrostatic fluid dynamics, its application to downscaled data-forced regional configurations—such as the North Adriatic Sea— is still limited by the lack of implementations of state of the art methods to accurately handle the open boundaries, mass conservation, and air-sea coupling.
This chapter details the mathematical and algorithmic interventions performed to bridge the gap between the library's core capabilities and the requirements of the Adriatic simulation. These modifications span the core solver (`Oceananigans.jl`) and the coupling interface (`ClimaOcean.jl`).

`Oceananigans.jl` provides both a `NonhydrostaticModel` and a `HydrostaticFreeSurfaceModel`. In principle, open boundary development could target either solver. The long-term objective of this work was to enable regional configurations from Northern Adriatic Sea to Mediterranean scale while retaining the capability to run high-resolution non-hydrostatic simulations.

However, during the initial phase of this work the non-hydrostatic solver did not yet provide the full infrastructure required for realistic regional configurations with data-forced open boundaries. In particular, several components of the boundary-condition workflow required for stable long integrations were still under active development in the Oceananigans codebase. For this reason, the hydrostatic free-surface solver was adopted as the development platform for the boundary-condition algorithms presented in this thesis.

The present work therefore constitutes a first hydrostatic open boundary prototype within Oceananigans. Importantly, the core elements of the boundary scheme developed here are solver-agnostic and can be transferred to the non-hydrostatic solver once the corresponding infrastructure becomes available. These elements include the local Orlanski phase-speed diagnostic, the multidimensional Raymond–Kuo radiation formulation, the CFL-limited phase-speed constraint, and the halo-based boundary update logic used to apply the boundary condition.

Before we present the developments we achieved, it is worth noting what the baseline of Oceananigans was before this work.

A method implementing relaxation of perturbations at the open boundary, named `PerturbationAdvection`, was developed as an open-boundary condition only in the

non-hydrostatic solver. It operates on the prognostic velocity components u, v, w and was designed under the assumption of independent velocity updates, without explicit coupling to a split free-surface barotropic solver.

Oceananigans baseline formulation corresponds to a 1D radiation condition acting only on normal propagation, while tangential components are ignored. The propagation speed is not diagnosed from the interior flow, but imposed externally. Consequently, oblique wave propagation is not represented, and no tangential gradient contribution is included.

Therefore, Oceananigans did not provide, prior to this thesis:

- A mechanism ensuring consistent mass-conserving barotropic adjustment at open boundaries, nor a framework for coupling and forcing by external depth-integrated (2D) fields.
- A ROMS-style RadNud implementation.
- A Chapman+Flather consistent barotropic open boundary.

The first two points are introduced and implemented in this work, whereas the third is not addressed.

3.1 Original state of solvers and OBCs in Oceananigans

3.1.1 Perturbation advection

The `PerturbationAdvection` boundary condition models the velocity near the open boundary as the sum of a slowly varying background flow and an internal disturbance,

$$u = U + u'. \tag{3.1}$$

The idea is that the large-scale velocity U represents the externally imposed mean transport, while the perturbation u' corresponds to disturbances generated inside the computational domain.

Under the assumptions that the background flow varies slowly near the boundary and that perturbations are small, the momentum equation reduces to a local advection–relaxation model for the perturbation. Expressed in terms of the full velocity field, the boundary evolution law becomes

$$\partial_t u \approx -U \cdot \nabla u' - \frac{u'}{\tau}. \tag{3.2}$$

Equation (3.2) states that disturbances are transported along the background flow U while being weakly relaxed toward the external state on a timescale τ. A detailed derivation of this reduced boundary model starting from the momentum equation is provided in the appendix A.2.

The boundary value of the diagnostic variable u from a backward Euler discretization for Eq. (3.2) expressed as:

$$u_i^{n+1} = \frac{u_i^n + \tilde{U}_i^{n+1} u_{i-1}^{n+1} + \tilde{\tau} U_i^{n+1}}{1 + \tilde{U}_i^{n+1} + \tilde{\tau}} \tag{3.3}$$

where $\tilde{U} = U\Delta t/\Delta x$ and $\tilde{\tau} = \Delta t/\tau$, are the dimensionless background velocity and nudging timescale, respectively, the boundary is considered here normal to the x axis, i is the coordinate of the boundary and $i-1$ is the coordinate of the adjacent interior point.

3.1.2 Momentum and free surface coupling in the hydrostatic solver

In a non-hydrostatic model, the momentum equation contains the full three-dimensional pressure gradient term $-\nabla p$, and the boundary velocity can be updated independently of the free-surface elevation. In contrast, the hydrostatic approximation replaces the pressure gradient by the barotropic term:

$$\nabla P = -g\nabla\eta,$$

where η is the free-surface elevation. This substitution has an important consequence for open boundary conditions: the boundary velocity is dynamically coupled to the free surface. In the hydrostatic regime, the depth-integrated (barotropic) momentum is governed by shallow-water dynamics, and the free surface controls the barotropic velocity. Therefore, any boundary condition applied to the velocity must remain consistent with the boundary condition applied to the free surface.

The corresponding discretized update for the boundary velocity u_i^{n+1}, obtained in the same manner as Eq. (3.3), is

$$u_i^{n+1} = \frac{u_i^n + \tilde{U}_i^{n+1}u_{i-1}^{n+1} + U_i^{n+1}\tilde{\tau}}{1 + \tilde{U}_i^{n+1} + \tilde{\tau}} - \frac{g \cdot (\frac{\Delta t}{\Delta x})}{1 + \tilde{U}_i^{n+1} + \tilde{\tau}}(\eta_i^{n+1} - \eta_{i-1}^{n+1}) \tag{3.4}$$

In the implementation developed during this thesis, the additional explicit hydrostatic contribution proportional to $g\nabla\eta$ expressed in equation 3.4 is not explicitly coded into the local boundary operations. The reason is that the free-surface field η^{n+1} is advanced globally through the barotropic continuity equation, using either the split-explicit procedure or the PCG solver, and the associated pressure correction is then applied to the velocity field separately [14]. Thus, the free-surface gradient contribution appearing in Eq. (3.4) is not treated by the local Orlanski update itself, but by the global barotropic solver.

It is important to note that the developments presented here do not replace the `PerturbationAdvection` boundary scheme itself. The implicit boundary update remains the same as in (3.3). The modification introduced in this work concerns only the background velocity U, which is no longer prescribed but dynamically diagnosed from the interior solution using the Orlanski phase–speed relation. In this way, the original scheme is preserved while allowing the radiation speed to adapt to the evolving flow. The extension to higher dimensions follows the same idea: the boundary update remains unchanged, while the diagnosis of the propagation velocity is generalized to include tangential gradients, leading to the Raymond–Kuo formulation used in the two–dimensional case.

3.2 New OBCs implemented

3.2.1 One-Dimensional Orlanski implementation

Current reasoning on the radiation speed follows the original Orlanski work [4]. Consider first the ideal one-dimensional outgoing-wave relation

$$\partial_t u + U\,\partial_x u = 0, \tag{3.5}$$

for a boundary variable u. In the formulation of `PerturbationAdvection`, U is assumed known a priori and represents the mean outward transport. Such an assumption is not

appropriate for a stable regional configuration. At an open boundary, the direction and magnitude of wave propagation are not fixed. They depend on the evolving interior solution. Prescribing a constant background velocity would therefore incorrectly constrain the radiation mechanism.

Specifically, we compute U as a local phase-speed estimated from the Orlanski radiation principle:

$$U = -\frac{\partial_t u}{\partial_x u},$$

using the ratio between temporal and spatial changes of the perturbation field at the adjacent interior points. This estimate is used only when it corresponds to outward propagation. If the signal indicates inflow, the radiative update is clapped to zero.

For test purposes, no external forcing or relaxation is included so that the behavior of the boundary is determined only by the radiation condition.

This diagnostic of the local phase-speed is not a pre-existing Oceananigans feature.

For the present velocity update, the background phase velocity U in Eq. (3.1) acts as this radiation speed and is updated dynamically from the perturbation field. Starting from

$$\partial_t u + u\nabla \cdot u = -\nabla \cdot P + F, \tag{3.6}$$

and neglecting the forcing term for the derivation, $F \approx 0$, one obtains after decomposition into background and perturbation parts

$$\partial_t u' + U\partial_x u' + \mathcal{O}(u'^2) = -(\partial_t U + U\partial_x U). \tag{3.7}$$

Under the usual assumption that the perturbation remains sufficiently small near the open boundary, the leading-order behaviour is therefore advective, with U playing the role of the radiation speed to be diagnosed from the grid-scale signal (see appendix A.1). For velocity, however, Orlanski radiation cannot be applied without additional admissibility conditions. Unlike tracers or free-surface height, the normal velocity can take either sign, so the diagnosed phase speed alone is not sufficient to define a stable and physically meaningful update. In particular, Orlanski supplements the discrete radiation condition with the following

$$\tilde{U} = \begin{cases} \dfrac{\Delta x}{\Delta t}, & \text{if } -\dfrac{\partial_t u'}{\partial_x u'} > \dfrac{\Delta x}{\Delta t}, \\ -\dfrac{\partial_t u'}{\partial_x u'}, & \text{if } 0 < -\dfrac{\partial_t u'}{\partial_x u'} < \dfrac{\Delta x}{\Delta t}, \\ 0, & \text{if } -\dfrac{\partial_t u'}{\partial_x u'} < 0. \end{cases} \tag{3.8}$$

Where the first condition is a CFL-type bound on the diagnosed propagation speed, ensuring that the numerical update remains stable; the second is a restriction to outward-directed propagation so that only disturbances leaving the domain are radiated; and the third is a safeguard for situations with weak mean flow, where a boundary that is initially inflow may later behave as outflow because of upstream-propagating disturbances generated inside the domain. These extra conditions are necessary to prevent the boundary update from interpreting incoming or weakly ambiguous signals as outgoing radiation.

In the present one-dimensional implementation, the analysis is first carried out without nudging in order to isolate the radiation mechanism itself and show, in the simplest setting, that an outgoing disturbance can leave the domain without artificial reflection. Once this

non-reflecting property is established for the pure radiation update, the addition of the nudging term is conceptually straightforward: it acts as an extra relaxation contribution toward the external state, without changing the basic outgoing-wave interpretation of the Orlanski part of the boundary condition.
We can write the discrete version of Eq. (3.8)

$$\tilde{U} = -\frac{u_{B-1}^{n+1} - u_{B-1}^{n}}{u_{B-1}^{n+1} - u_{B-2}^{n+1}} \tag{3.9}$$

where B identifies the coordinate of the boundary point, $B-1$ is the coordinate of the adjacent point, and $B-2$ the coordinate of the interior point.
Based on these requirements, we now present the algorithmic formulation of the perturbation advection open boundary condition. Algorithm (1) summarizes the sequence of operations used at each time step to diagnose the local phase speed, enforce stability constraints, and update the boundary velocity accordingly.

Algorithm 1 Perturbation Advection Open Boundary Condition

procedure UPDATEBOUNDARY($u_i^n, u_{i-1}^{n+1}, u_{i-2}^{n+1}, \Delta t, \Delta X$)
▷ u_i: boundary, u_{i-1}: adjacent node, u_{i-2}: interior node
  $\Delta u_t \leftarrow u_{i-1}^{n+1} - u_{i-1}^{n}$ ▷ Temporal gradient at adjacent cell
  $\Delta u_x \leftarrow u_{i-1}^{n+1} - u_{i-2}^{n+1}$ ▷ Spatial gradient in the interior
  if $|\Delta u_x \cdot \Delta t| > \epsilon$ **then** ▷ Dimensionless phase speed ratio
    $\tilde{U} \leftarrow -\frac{\Delta u_t}{\Delta u_x}$
  else
    $\tilde{U} \leftarrow 0$
  end if
  $\tilde{U}_{max} \leftarrow 0.49\sqrt{gH}\frac{\Delta t}{\Delta X}$ ▷ Stability limit (shallow water dimensionless speed)
  $\tilde{U} \leftarrow \max(0, \min(\tilde{U}_{max}, \tilde{U}))$ ▷ Constrain to outward advection
  $u_i^{n+1} \leftarrow \frac{u_i^n + \tilde{U} u_{i-1}^{n+1}}{1+\tilde{U}}$ ▷ Implicit backward Euler update
  return u_i^{n+1}
end procedure

As discussed above, the perturbation advection scheme provides a consistent and stable boundary update in the one-dimensional setting, where propagation is assumed to be normal to the open boundary. This assumption becomes problematic when extending the method to two- or three-dimensional flows. In such cases, disturbances often approach the boundary obliquely, and diagnosing the phase speed using only the normal gradient leads to a systematic underestimation of the true outward propagation speed, as described in Section 2.2. This underestimation increases numerical reflection and can compromise stability in realistic regional configurations.

In addition, pure radiation based on locally diagnosed phase speeds is insufficient to constrain the solution during sustained inflow events. As emphasized in Section 2.3.1, long-term stability in data-forced regional models requires a relaxation mechanism toward external fields, with a timescale that adapts to the direction of propagation. These limitations motivate the extension of the present formulation to higher dimensions and the incorporation of a radiation–nudging strategy.

3.2.2 Two (or Three)-Dimensional Raymond–Kuo generalization

In two spatial dimensions, disturbances generally approach an open boundary with oblique incidence. If one diagnoses a radiation speed using only the normal gradient, the one-dimensional normal-only diagnostic becomes incomplete for oblique incidence and ill-conditioned when the normal gradient is small. This causes tangential propagation to be misdiagnosed as normal propagation and increases reflection. A practical remedy is to use a two-dimensional Orlanski-type formulation, following the Raymond–Kuo generalization (2.2), where the diagnosis incorporates both normal and tangential gradients.

Discrete two-dimensional radiation update in Marchesiello et al. [11] Marchesiello formulate a two-dimensional radiation condition for ROMS by discretizing the multidimensional relation already introduced equation 2.6:

$$\partial_t u + c_x \partial_x u + c_e \partial_e u = 0, \tag{3.10}$$

where e denotes the outward coordinate to the boundary and x the tangential coordinate along the boundary. After finite-difference discretization and multiplication by Δt, the equation is written in dimensionless form

$$r_x := \frac{c_x \Delta t}{\Delta x}, \qquad r_e := \frac{c_e \Delta t}{\Delta e},$$

For the south boundary considered here, whose outward normal points in the $+y$ direction, we rewrite their update directly in the halo notation used in the present implementation: the boundary value is stored at $j = j_{\text{halo}}$, the first interior point at $j_{\text{adj}} = j_{\text{halo}} + 1$, and the second interior point at $j_{\text{intr}} = j_{\text{halo}} + 2$. With this identification, the discretized Marchesiello south-boundary field update reads

$$(1 + r_e)\, u^{n+1}_{i,j_{\text{halo}},k} = u^{n}_{i,j_{\text{halo}},k} + r_e\, u^{n+1}_{i,j_{\text{adj}},k} - r_x\, g^{-}_{\text{halo}}, \qquad r_x > 0, \tag{3.11}$$

$$(1 + r_e)\, u^{n+1}_{i,j_{\text{halo}},k} = u^{n}_{i,j_{\text{halo}},k} + r_e\, u^{n+1}_{i,j_{\text{adj}},k} - r_x\, g^{+}_{\text{halo}}, \qquad r_x < 0, \tag{3.12}$$

where the tangential boundary gradients are

$$g^{-}_{\text{halo}} := u^{n}_{i,j_{\text{halo}},k} - u^{n}_{i-1,j_{\text{halo}},k}, \qquad g^{+}_{\text{halo}} := u^{n}_{i+1,j_{\text{halo}},k} - u^{n}_{i,j_{\text{halo}},k}. \tag{3.13}$$

and the next discrete coefficients, corresponding to c_e and c_x previously defined equation 2.7, are defined as:

$$r_e := -\frac{\Delta u_t\, \Delta u_y}{(\Delta u_x)^2 + (\Delta u_y)^2}, \qquad r_x := -\frac{\Delta u_t\, \Delta u_x}{(\Delta u_x)^2 + (\Delta u_y)^2}, \tag{3.14}$$

The local temporal and normal differences are evaluated at the adjacent interior point,

$$\Delta u_t := u^{n+1}_{i,j_{\text{adj}},k} - u^{n}_{i,j_{\text{adj}},k}, \qquad \Delta u_y := u^{n+1}_{i,j_{\text{adj}},k} - u^{n+1}_{i,j_{\text{intr}},k}, \tag{3.15}$$

The tangential difference Δu_x is selected by an upwind rule at the adjacent interior line $j = j_{\text{adj}}$. The tangential difference is defined by the authors through the upwind rule

$$\Delta u_x := \begin{cases} g^{-}_{\text{adj}}, & \text{if } \Delta u_t \left(g^{-}_{\text{adj}} + g^{+}_{\text{adj}} \right) > 0, \\ g^{+}_{\text{adj}}, & \text{otherwise.} \end{cases} \tag{3.16}$$

using one-sided tangential gradients at the adjacent interior line:

$$g^{-}_{\mathrm{adj}} := u^{n}_{i,j_{\mathrm{adj}},k} - u^{n}_{i-1,j_{\mathrm{adj}},k}, \tag{3.17}$$

$$g^{+}_{\mathrm{adj}} := u^{n}_{i+1,j_{\mathrm{adj}},k} - u^{n}_{i,j_{\mathrm{adj}},k}, \tag{3.18}$$

Thus the tangential derivative entering the radiation diagnostic is not centered, but chosen so as to remain consistent with the locally inferred propagation direction. In this form, the boundary value is updated from the previous boundary state, the newly available adjacent interior value, and an upwinded tangential correction whose sign depends on the diagnosed tangential propagation.

ROMS implementation form and sign convention. The form actually implemented in ROMS is algebraically equivalent to the previous update, but it is written in terms of coefficients that are more convenient for a halo-based numerical kernel. This is also the form adopted in Oceananigans. Define first

$$C_{\mathrm{ff}} := (\Delta u_x)^2 + (\Delta u_y)^2, \tag{3.19}$$

$$C_e := \Delta u_{-t}\, \Delta u_y, \tag{3.20}$$

and, when the two-dimensional radiation option is enabled,

$$C_x := \min(C_{\mathrm{ff}}, \max(\Delta u_{-t}\Delta u_x, -C_{\mathrm{ff}}))\,, \tag{3.21}$$

while otherwise $C_x := 0$.

Here, however, as adopted in the ROMS implementation convention, the temporal difference is written with a change of sign and includes an additional practical safeguard when $\Delta u_{-t}\Delta u_y < 0$ that is not written explicitly in the analytic form of Marchesiello et al. :

$$\Delta u_{-t} := \begin{cases} u^{n}_{i,j_{\mathrm{adj}},k} - u^{n+1}_{i,j_{\mathrm{adj}},k} = -\,\Delta u_t & \text{if} \quad \Delta u_{-t}\Delta u_y > 0 \\ 0 & \text{if} \quad \Delta u_{-t}\Delta u_y < 0 \end{cases} \tag{3.22}$$

For $\Delta u_{-t}\Delta u_y > 0$ the sign change is purely conventional: it does not define a different boundary condition, but only rewrites the same Marchesiello diagnostic in the convention used in the ROMS source code and followed here in Oceananigans. The safeguard for $\Delta u_{-t}\Delta u_y < 0$ removes locally inconsistent signals, which would otherwise be interpreted as spurious inflow by the radiation diagnostic.

With this notation,

$$r_e = \frac{C_e}{C_{\mathrm{ff}}}, \qquad r_x = \frac{\Delta u_{-t}\Delta u_x}{C_{\mathrm{ff}}},$$

which coincide identically with the Marchesiello diagnostic.

ROMS/Oceananigans boundary update. With the previous definitions, the implemented south-boundary update at the halo line $j = j_{\mathrm{halo}}$ is

$$u^{n+1}_{i,j_{\mathrm{halo}},k} = \frac{C_{\mathrm{ff}}\, u^{n}_{i,j_{\mathrm{halo}},k} + C_e\, u^{n+1}_{i,j_{\mathrm{adj}},k} - \max(C_x, 0)\, g^{-}_{\mathrm{halo}} - \min(C_x, 0)\, g^{+}_{\mathrm{halo}}}{\max(C_{\mathrm{ff}} + C_e, \epsilon_{\mathrm{O}})}, \tag{3.23}$$

where $\epsilon_{\mathrm{O}} > 0$ is a small positive constant preventing division by zero. It is worth noting that the denominator in Eq. (3.23) is not a new algebraic choice introduced here. It is part of the discrete ROMS form itself that we adapt for Oceananigans.

Equation (3.23) shows that the boundary update is entirely controlled by the coefficients C_{ff}, C_e, and C_x, which act as weights in the blending of halo, interior, and tangential contributions. Their roles can therefore be interpreted directly as follows:

- $C_{\text{ff}} = (\Delta u_x)^2 + (\Delta u_y)^2$ is a local "energy" scale for the 2D gradient of the adjacent interior signal. When the interior gradients vanish, C_{ff} is small and the scheme automatically reduces sensitivity to the tangential correction.
- $C_e = \Delta u_t \, \Delta u_y$ acts as the normal-radiation coupling term. If the temporal change Δu_t and the normal gradient Δu_y are consistent with outward propagation, then C_e is positive and the boundary is pulled toward the adjacent interior value.
- C_x sets the strength and sign of tangential advection along the boundary. The min/max bounding in (3.14) prevents C_x from exceeding the scale set by C_{ff}, which is a practical way to avoid over-correction when tangential gradients dominate because of noise or near-uniform flow.

This coefficient form is close in spirit to ROMS-style multidimensional radiation updates: it uses interior history to choose a direction and then blends boundary and adjacent values with an upwinded tangential term.

Directional nudging toward external data. Pure radiation is insufficient during sustained inflow, and long-term stability in data-forced regional integrations requires relaxation toward external boundary fields (cf. Section 2.3.1). After the radiation step, we classify the boundary regime using

$$\sigma := \Delta u_{-t} \Delta u_y, \tag{3.24}$$

and choose a nudging timescale

$$\tau := \begin{cases} \tau_{\text{in}}, & \text{if } \sigma < 0, \\ \tau_{\text{out}}, & \text{otherwise.} \end{cases} \tag{3.25}$$

Let u_{ext} be the prescribed exterior value. The implementation supports two nudging variants.
In the implicit variant, consistent with the common ROMS formulation, define

$$\phi := \frac{\Delta t}{\tau + \Delta t}, \tag{3.26}$$

and update

$$u^{n+1} \leftarrow (1 - \phi)\, u_{\text{rad}}^{n+1} + \phi\, u_{ext}. \tag{3.27}$$

where u_{rad}^{n+1} is the updated boundary velocity that is obtained by applying Eq. (3.23).
In the explicit-increment variant, we define

$$\tilde{\tau} := \frac{\Delta t}{\tau}, \tag{3.28}$$

and apply

$$u^{n+1} \leftarrow u_{\text{rad}}^{n+1} + \tilde{\tau}\, (u_{ext} - u_{\text{halo}}^{n}), \tag{3.29}$$

where u^n_{halo} denotes the boundary value from the previous time level.
Finally, to overcome some instabilites we apply a CFL limiter on the boundary value,

$$\text{CFL} := \frac{|u^{n+1}|\,\Delta t}{\Delta X}, \tag{3.30}$$

and if $\text{CFL} > \text{CFL}_{\max}$ we rescale $u^{n+1} \leftarrow u^{n+1}\,(\text{CFL}_{\max}/\text{CFL})$.

This two-dimensional formulation (Algorithm 2) extends the one-dimensional perturbation advection scheme by explicitly accounting for oblique wave propagation through the inclusion of tangential gradients in the phase-speed diagnosis.

Algorithm 2 Two-Dimensional Perturbation Advection Open Boundary Condition

procedure UPDATEBOUNDARY2D($u^{n+1}, u^n, (i, j_{\text{halo}}, k), \Delta t, \Delta X, u_{ext}, \tau_{\text{in}}, \tau_{\text{out}}$)
▷ u^{n+1}: current time level, u^n: previous time level
  $j_{\text{adj}} \leftarrow j_{\text{halo}} + 1, \quad j_{\text{intr}} \leftarrow j_{\text{halo}} + 2$
  ▷ 1. Temporal and normal differences at adjacent interior point
  $\Delta u_{-t} \leftarrow u^n_{i,j_{\text{adj}},k} - u^{n+1}_{i,j_{\text{adj}},k}$
  $\Delta u_y \leftarrow u^{n+1}_{i,j_{\text{adj}},k} - u^{n+1}_{i,j_{\text{intr}},k}$
  if $\Delta u_{-t} \cdot \Delta u_y < 0$ **then**
    $\Delta u_{-t} \leftarrow 0$
  end if
  ▷ 2. Tangential gradients at j_{adj} (previous time level)
  $g^-_{\text{adj}} \leftarrow u^n_{i,j_{\text{adj}},k} - u^n_{i-1,j_{\text{adj}},k}$
  $g^+_{\text{adj}} \leftarrow u^n_{i+1,j_{\text{adj}},k} - u^n_{i,j_{\text{adj}},k}$
  if $\Delta u_{-t} \cdot (g^-_{\text{adj}} + g^+_{\text{adj}}) > 0$ **then**
    $\Delta V_x \leftarrow g^-_{\text{adj}}$
  else
    $\Delta V_x \leftarrow g^+_{\text{adj}}$
  end if
  ▷ 3. Radiation coefficients
  $\text{cff} \leftarrow (\Delta V_x)^2 + (\Delta u_y)^2$
  $C_e \leftarrow \Delta u_{-t} \cdot \Delta u_y$
  $C_x \leftarrow \min(\text{cff}, \max(\Delta u_{-t} \cdot \Delta V_x, -\text{cff}))$
  $\text{denom} \leftarrow \max(\text{cff} + C_e, \epsilon_{\text{O}})$
  ▷ 4. Tangential gradients at halo line j_{halo} (previous time level)
  $g^-_{\text{halo}} \leftarrow u^n_{i,j_{\text{halo}},k} - u^n_{i-1,j_{\text{halo}},k}$
  $g^+_{\text{halo}} \leftarrow u^n_{i+1,j_{\text{halo}},k} - u^n_{i,j_{\text{halo}},k}$
  ▷ 5. Two-dimensional radiation update at boundary
  $u^{n+1}_{\text{rad}} \leftarrow \dfrac{\text{cff}\, u^n_{i,j_{\text{halo}},k} + C_e\, u^{n+1}_{i,j_{\text{adj}},k} - \max(C_x, 0)\, g^-_{\text{halo}} - \min(C_x, 0)\, g^+_{\text{halo}}}{\text{denom}}$
  $u^{n+1}_{i,j_{\text{halo}},k} \leftarrow u^{n+1}_{\text{rad}}$
  if $\Delta u_{-t} \cdot \Delta u_y < 0$ **then** ▷ 6. Select nudging timescale using $\sigma = \Delta u_{-t} \Delta u_y$
    $\tau \leftarrow \tau_{\text{in}}$
  else
    $\tau \leftarrow \tau_{\text{out}}$
  end if
  ▷ 7. Apply nudging toward exterior value if τ is finite and positive
  if $\tau > 0$ **and** $\tau < \infty$ **and** $\Delta t > 0$ **then**
    $\phi \leftarrow \dfrac{\Delta t}{\tau + \Delta t}$ ▷ Implicit (or exterior) nudging
    $u^{n+1}_{i,j_{\text{halo}},k} \leftarrow (1 - \phi)\, u^{n+1}_{i,j_{\text{halo}},k} + \phi\, u_{ext}$
  end if
  ▷ 8. CFL limiter at the boundary
  $\text{CFL} \leftarrow \dfrac{|u^{n+1}_{i,j_{\text{halo}},k}|\, \Delta t}{\Delta X}$
  if $\text{CFL} > \text{CFL}_{\max} = 0.3$ **then**
    $u^{n+1}_{i,j_{\text{halo}},k} \leftarrow u^{n+1}_{i,j_{\text{halo}},k} \dfrac{\text{CFL}_{\max}}{\text{CFL}}$
  end if
  return $u^{n+1}_{i,j_{\text{halo}},k}$
end procedure

3.3 Core Solver Improvments

3.3.1 Handling bidimensional Fields at Boundaries

A specific issue arose when applying boundary conditions to fields with a reduced number of dimensions, such as the barotropic free surface $\eta(x, y)$, which lacks a vertical z-dimension. The default `getbc` methods in Oceananigans assume a 3D field structure, leading to indexing errors when accessing 2D `XYReducedField` objects.
To resolve this, a specialized dispatch was added to `field.jl`. This ensures that boundary conditions for the free surface are correctly retrieved without attempting to index into a non-existent vertical dimension.
Implementation in `field.jl`:

```
@inline getbc(condition::XYReducedField,
              ::Integer,
              k::Integer,
              ::AbstractGrid,
              args...) =
    @inbounds condition[1, 1, k]
```

3.3.2 Mass Conservation for Hydrostatic Model

In regional hydrostatic models, small numerical errors in open boundary velocities can integrate over time to cause a net gain or loss of mass, leading to an unrealistic drift in the mean sea level. While `NonhydrostaticModel` had provisions for tracking boundary mass flux, the `HydrostaticFreeSurfaceModel` —which is required for the Adriatic simulation—did not.
To fix this, the `HydrostaticFreeSurfaceModel` struct was extended to include a `boundary_mass_fluxes` container. This allows the solver to track the net volume transport entering and leaving the domain at every timestep.
Modification to `hydrostatic_free_surface_model.jl`:

```
mutable struct HydrostaticFreeSurfaceModel{TS, E, A<:AbstractArchitecture, S
                                           G, T, V, B, R, F, P, BGC, U, C, $
    #... [existing fields]...
    auxiliary_fields :: AF      # User-specified auxiliary fields

    # Track boundary mass fluxes for open boundaries
    boundary_mass_fluxes :: BM  # Container for averaged mass fluxes at open

    vertical_coordinate :: Z    # Rulesets that define the time-evolution of
end

#... inside the constructor...
boundary_mass_fluxes = initialize_boundary_mass_fluxes(velocities)
```

This structural change was coupled with a dynamical update step. In the time-stepping loop, a call to `enforce_open_boundary_mass_conservation!` was injected. This function calculates the net divergence across all open boundaries and adds a uniform barotropic

correction velocity to ensure that:

$$\oint_{\partial\Omega} \mathbf{u} \cdot \mathbf{n}\, dA = \int_{\text{surface}} (E - P)\, dA + \sum Q_{\text{rivers}} \tag{3.31}$$

Modification to `update_hydrostatic_free_surface_model_state.jl`:

```
    import Oceananigans.Models.NonhydrostaticModels: enforce_open_boundary_m
#...
# Enforce open boundary mass conservation after updating BCs
enforce_open_boundary_mass_conservation!(model, model.boundary_mass_fluxes)
```

Here we present the extension of the mass-conserving algorithm, originally formulated for the non-hydrostatic model, to the hydrostatic case. The same construction applies without modification, ensuring mass conservation in both hydrostatic and non-hydrostatic models (Algorithm 3.3.2).

Algorithm 3 Mass Conservation Correction for Open Boundaries

Require: Model velocities $\mathbf{u}$ and boundary flux structure M
Ensure: Net mass flux $\sum \Phi = 0$

Update Fluxes: For each open boundary side $s \in$ West, East, South, North, Bottom, Top, update $\Phi_s = \iint_{A_s} u_n, dA_s$.
Compute Net Imbalance:
  $\Delta\Phi \leftarrow (\Phi_{west} + \Phi_{south} + \Phi_{bottom}) - (\Phi_{east} + \Phi_{north} + \Phi_{top})$
Calculate Global Area:
  $A_{total} \leftarrow \sum \text{Area}(s)$ for all boundaries participating in the correction.
Determine Correction Speed:
  $u_{corr} \leftarrow \Delta\Phi / A_{total}$
Apply Correction:
for each participating boundary s **do**
  if $s \in$ West, South, Bottom **then**
    $u_n|s \leftarrow u_n|s - u_{corr}$
  else ▷ $s \in$ East, North, Top
    $u_n|s \leftarrow u_n|s + u_{corr}$
  end if
end for

3.3.3 Thermodynamic Consistency in Atmospheric Coupling

Accurate heat flux calculation is critical for capturing the cooling effects of the wind. The JRA55-do atmospheric reanalysis dataset provides air temperature in degrees Celsius, while the bulk aerodynamic formulas in `ClimaOcean`'s `atmosphere_ocean_fluxes.jl` assumed Kelvin for specific radiative terms. This unit mismatch can lead to erroneous latent and sensible heat fluxes ($Q_h \propto \Delta T$), and catastrophic errors in longwave radiation ($Q_b \propto T^4$).
To correct this, a unit handling system was implemented. A `temperature_units` parameter was added to the atmosphere interface, and a conversion step was injected into the flux computation kernel.
Modification to `atmosphere_ocean_fluxes.jl`:

```
function compute_atmosphere_ocean_fluxes!(coupled_model)
    #...
    atmosphere_properties = (
        #...
        temperature_units = coupled_model.interfaces.properties.atmosphere_t
    )
    #...
end

# Inside the kernel:
T_a = convert_to_kelvin(temperature_units, atmosphere_state.T[i, j, 1])
```

3.3.4 Modular Lateral Boundary Configuration

Finally, the setup script `ocean_simulation.jl` was refactored to allow for more granular control over lateral boundary conditions. The original implementation struggled with the positional arguments required by the `FieldBoundaryConditions` constructor when mixing default and user-specified conditions.
The code was modified to explicitly parse user inputs for `u`, `v`, `T`, and `S` separately, allowing for a hybrid configuration where, for example, velocity boundaries are radiative (Orlanski) while tracer boundaries are clamped or nudged.
Modification to `ocean_simulation.jl`:

```
    # Gather user-supplied lateral BCs
user_u = get(lateral_boundary_conditions, :u, NamedTuple())
user_v = get(lateral_boundary_conditions, :v, NamedTuple())
user_T = get(lateral_boundary_conditions, :T, NamedTuple())
user_S = get(lateral_boundary_conditions, :S, NamedTuple())

# Helper to build positional FieldBoundaryConditions safely
mk_fbc(user_lat, top_bc, bottom_bc, immersed_bc) = FieldBoundaryConditions(
    get(user_lat, :west,  default_bc()),
    get(user_lat, :east,  default_bc()),
    get(user_lat, :south, default_bc()),
    get(user_lat, :north, default_bc()),
    bottom_bc,
    top_bc,
    immersed_bc,
)

# Explicit construction avoids positional errors
boundary_conditions = (
    u = mk_fbc(user_u, u_top_bc, u_bot_bc, u_immersed_bc),
    #......
)
```

3.3.5 Summary of Implemented Boundary Features

The present implementation includes:

- A perturbation-advection radiation scheme for the boundary-normal velocity component.
- A two-dimensional phase-speed diagnostic including tangential gradients.
- CFL limiting of diagnosed radiation speeds.
- Mass-flux correction for the hydrostatic solver to reduce long-term volume drift.

The present implementation does not include a full Flather condition coupling depth-integrated velocity to sea surface elevation differences.

The scheme should therefore be interpreted as a methodological prototype for hydrostatic velocity radiation rather than a complete operational open boundary system.

Chapter 4

Numerical Results and Validation

This chapter presents the first numerical results obtained with the perturbation advection open boundary condition introduced in Chapter 3. The goal is validate the two previous algorithms 1-2.

4.1 One-Dimensional System

The simulation that is described here uses Algorithm 1. At each time step, the boundary update proceeds by diagnosing a local phase speed from interior values, constraining this speed to enforce outward propagation and CFL stability, and applying an implicit backward Euler update at the boundary point.
The algorithm operates purely on local information near the boundary and does not require any externally prescribed phase speed. This property makes it well suited for idealized tests where the objective is to verify that internally generated disturbances exit the computational domain without reflection or artificial amplification.

4.1.1 Model Configuration

The simulation uses the `HydrostaticFreeSurfaceModel` from `Oceananigans.jl` on a rectilinear grid. The domain is one–dimensional in the horizontal direction with a shallow vertical extent, intended to isolate barotropic dynamics.
The hydrostatic approximation is justified because the horizontal scale, L_x, of the perturbation ($\mathcal{O}(10-100)$km) is several orders of magnitude larger than the water depth, $H \sim (10\text{m})$, yielding a ratio, $H/L_x \sim 10^{-5}$. Under shallow water scaling, vertical accelerations are $\mathcal{O}((H/L_x)^2)$ relative to gravity and therefore negligible. The dynamics are dominated by long barotropic gravity waves, for which pressure is determined by the hydrostatic weight of the water column. Consequently, the `HydrostaticFreeSurfaceModel` provides a consistent asymptotic description of the problem.
The horizontal grid consists of 50 points spanning a domain of length

$$L_x = 500\,\text{km}$$

with bounded topology at both west and east boundaries. The vertical coordinate spans from $z = -10\,\text{m}$ to the free surface at $z = 0$, and no vertical dynamics are resolved beyond the hydrostatic balance.
Open boundary conditions are applied to the zonal velocity component u at both the

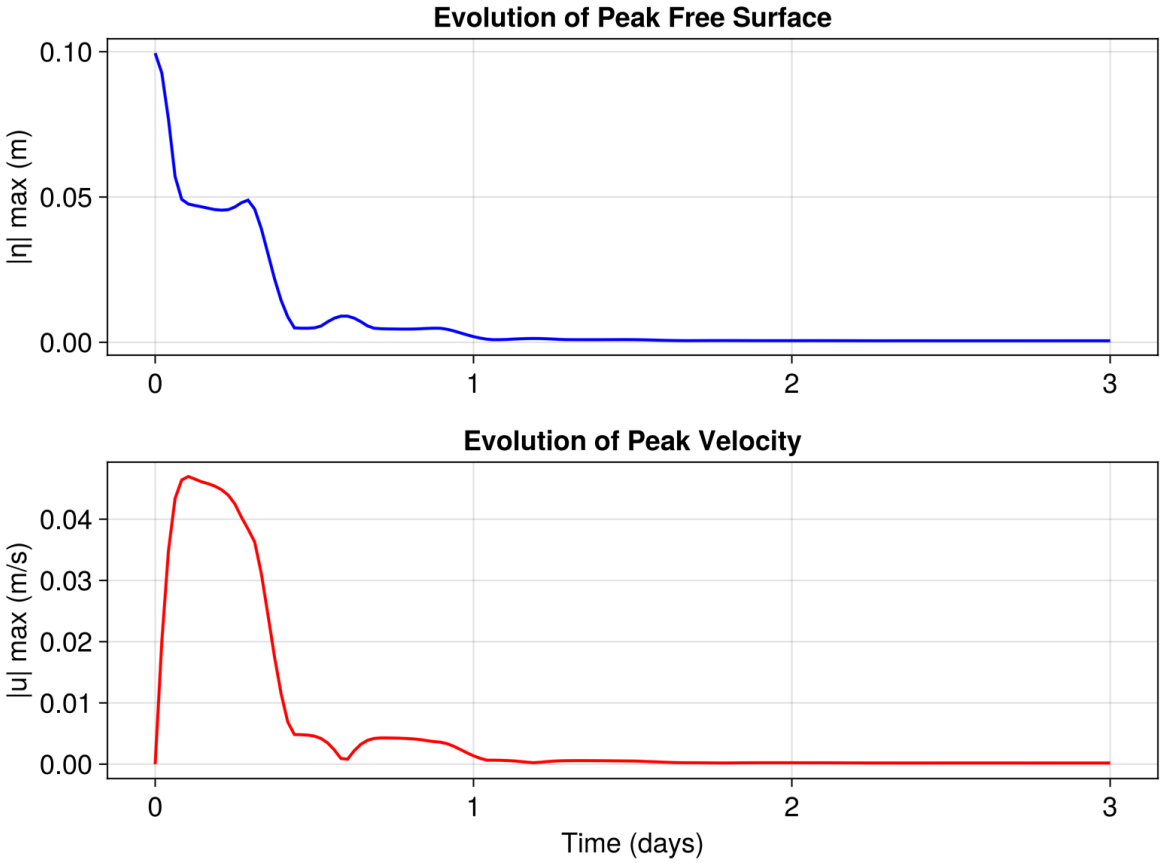

Figure 4.1: Time evolution of the maximum absolute free surface elevation $|\eta|_{\max}$ (top) and maximum absolute velocity $|u|_{\max}$ (bottom) in the one–dimensional experiment. Both quantities decay to zero as the perturbation propagates out of the domain, indicating effective radiation of barotropic disturbances through the open boundaries.

eastern and western boundaries using the `PerturbationAdvection` scheme. No external velocity is prescribed. This choice ensures that any motion at the boundary results solely from interior dynamics rather than imposed forcing.
The free surface is treated implicitly through the barotropic solver. The initial condition consists of a localized Gaussian perturbation in the free surface elevation,

$$\eta_0(x) = 0.1 \exp\Big(- \frac{(x - x_0)^2}{2\sigma_x^2} \Big) \tag{4.1}$$

with center $x_0 = 250\,\text{km}$ and width $\sigma_x = 50\,\text{km}$. This perturbation generates gravity wave motion that propagates toward both open boundaries.
The time step is fixed to $\Delta t = 5\,\text{minutes}$, minutes, and the simulation is run for a total of 3 days. Diagnostic outputs for free surface elevation η, velocity u, and barotropic volume flux are written every 30 minutes.

4.1.2 Evolution of Free Surface and Velocity

Figure 4.1 shows the time evolution of the maximum absolute free surface elevation $|\eta|_{\max}$ and the maximum absolute velocity $|u|_{\max}$ over the entire domain.

At early times, the initial perturbation generates a clear barotropic response. Following initialization, the maximum sea-surface height $|\eta|_{\max}$ decreases rapidly during the early adjustment phase, after which it reaches a quasi-steady plateau while the wave propagates toward the open boundary. The leading wave packet arrives at the boundary at approximately day 0.3, consistent with the expected shallow-water phase speed $\sqrt{gh} \simeq 10 m.s^{-1}$ Upon reaching the open boundary, the maximum SSH drops abruptly to roughly 10% of its pre-boundary value, indicating that the boundary condition effectively allows outgoing wave energy to exit the domain. Only a weak reflected signal is generated, which propagates back across the domain over the following 0.6 days. This reflected wave subsequently reaches the opposite boundary, where it is almost completely dissipated, with negligible remaining energy.
In the meanwhile, the maximum velocity $|u|_{\max}$ increase rapidly as the initial Gaussian free-surface perturbation adjusts and is converted into propagating gravity waves. During this adjustment phase, part of the available potential energy stored in the surface elevation anomaly is transferred into kinetic energy, leading to a sharp rise in the domain-wide maximum velocity.

Once the wave field is established, the maximum velocity remains high while the wave packet propagates across the interior of the domain. When the leading wave reaches the open boundary (around day 0.3), the maximum velocity drops abruptly, mirroring the sharp reduction observed in the maximum SSH. This decrease reflects the efficient radiation of kinetic energy out of the domain through the open boundary. A weak reflected signal produces a secondary, much smaller peak in the maximum velocity as it propagates back across the domain. This reflected velocity signal decays in time, and when it reaches the opposite boundary it is almost entirely canceled, leaving only negligible residual velocities in the domain.
After approximately one day, the maxima approach values close to numerical zero and remain there for the remainder of the simulation. No secondary growth, oscillations, or standing wave patterns are observed.
This behavior admits a direct physical interpretation. The decay of $|\eta|_{\max}$ indicates that the free surface perturbation exits the computational domain. The concurrent decay of $|u|_{\max}$ shows that the associated momentum also leaves the domain rather than being reflected back into the interior.
Since no explicit dissipation or interior damping is applied, this decay cannot be attributed to numerical diffusion. It demonstrates that the perturbation advection boundary condition successfully radiates outgoing barotropic disturbances.

4.1.3 Numerical Stability

The diagnosed boundary phase speed is constrained by a maximum admissible value $U_{\max}$, defined in Algorithm 1 as a fraction of the shallow water gravity wave speed,

$$U_{\max} = \alpha\sqrt{gH},$$

with $\alpha < 1$. In the present experiments, the choice $\alpha = 0.49$ ensures that the boundary Courant number remains strictly below unity.

This parameter is not fixed by theory. It represents a numerical stability constraint that must be selected by the user. Larger values of $U_{\max}$ reduce boundary damping but increase the risk of instability. Smaller values increase robustness at the cost of stronger attenuation.

The results presented here show that the chosen value yields a stable and reflection–free solution for this configuration. The optimal choice of $U_{\max}$ remains problem dependent and must be tuned when changing resolution, depth, or forcing.

4.2 Two-Dimensional System: North Adriatic Sea

4.2.1 Simulation setup

The simulation that is described here uses Algorithm 2, but is embedded in a realistic geometry, forced by external reanalysis data, and coupled to atmospheric fluxes.
The horizontal grid was provided by OGS for the North-Adriatic simulations at 1/128 th degree resolution, on a domain of size 494×300 in the zonal and meridional directions. The vertical coordinate uses a stretched Z-level discretization made of 60 levels whose thickness varies from 0.5 meters at the surface to about 20 meters at depth and includes an immersed boundary representation of the bathymetry made by depth-dependent land-masked cells.

To reduce computational cost during the development and validation phase, we also performed simulations on a coarsened configuration (horizontal resolution $1/12$, 15 vertical levels). This setup was used solely to verify the numerical implementation and assess qualitative consistency of the results before going into full-resolution configuration.

Open boundary forcing is imposed only at the southern boundary of the regional domain. The meridional velocity v is treated with an `OpenBoundaryCondition` using the `PerturbationAdvection` scheme. By contrast, the zonal velocity u, temperature T, and salinity S are not updated through a radiative scheme in this experiment. They are prescribed directly from GLORYS reanalysis fields through time-dependent discrete boundary functions wrapped in `ValueBoundaryCondition`, so that their southern boundary values are imposed in Dirichlet form from the external dataset. Atmospheric forcing is provided by a prescribed JRA55 configuration, and the ocean free surface is advanced with the `ImplicitFreeSurface` solver.

4.2.2 Stability limitations

The simulation is integrated with a fixed time step $\Delta t = 60$ seconds and proceeds well for the first two days, while, the advective CFL number is actively monitored and remains bounded by

$$\mathrm{CFL} \approx 6 \times 10^{-3},$$

which is well below standard stability thresholds for explicit advection schemes. However, after two days of simulation non-stable CFL values suddenly appear, more on this will be discussed in the next section.

4.2.3 Time Evolution of Tracers and Velocity Magnitudes

We present here the simulation for the first 2 days of simulation time.
Figures 4.2 and 4.3 show the time evolution of the maximum absolute salinity and temperature magnitudes over the computational domain. Both tracers evolve smoothly in time, with no abrupt jumps or oscillations prior to model termination.
Salinity exhibits a slow monotonic decrease in its maximum value, consistent with advective

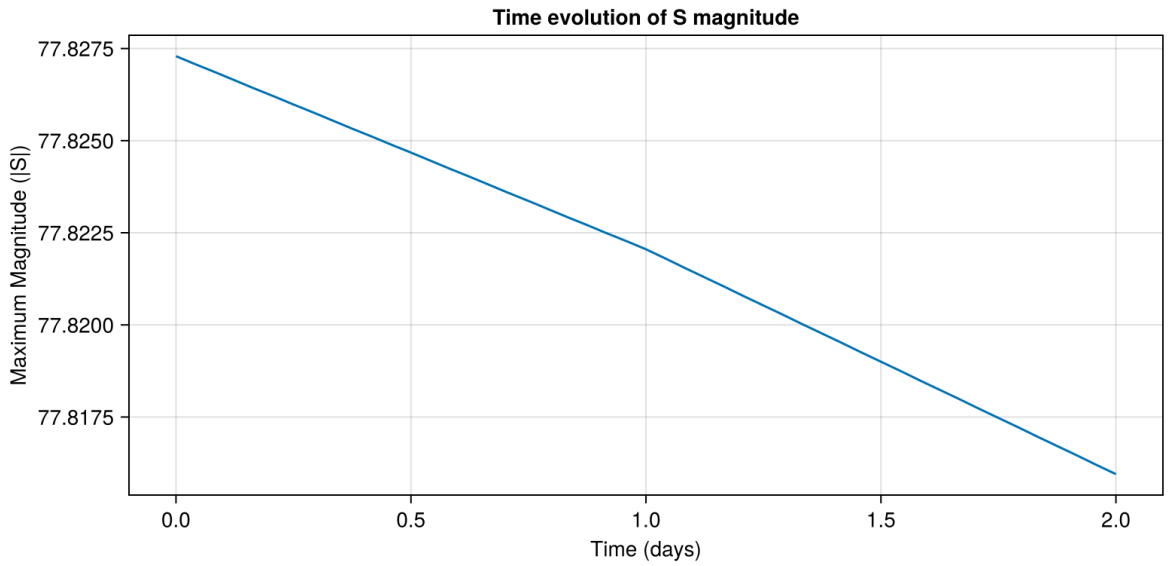

Figure 4.2: Time evolution of the maximum absolute salinity magnitude $|S|_{\max}$ in the two–dimensional Adriatic simulation. The signal evolves smoothly in time without abrupt excursions prior to model termination.

redistribution and boundary exchange rather than numerical instability. Temperature shows a similar gradual evolution, with changes occurring over long time scales relative to the barotropic dynamics.
Figures 4.4, 4.5, and 4.6 show the evolution of the maximum absolute velocity components. The zonal velocity u remains bounded throughout the simulation window, with moderate growth followed by partial relaxation. The meridional velocity v increases more substantially, reflecting the imposed southern inflow and the dominant along–basin circulation typical of the Adriatic configuration. The vertical velocity w remains small compared to horizontal components, as expected in a hydrostatic regime, though it shows coherent growth associated with resolved vertical motions.
Crucially, none of these diagnostics exhibit the rapid exponential growth characteristic of CFL violation or explicit advection instability for the first two days of simulation.

4.2.4 Model Termination and NaN Emergence

Despite the stable behavior observed in global diagnostics, the simulation eventually terminates due to the emergence of non–finite values (NaNs) in the velocity fields. These NaNs are detected explicitly through runtime callbacks that scan the velocity arrays at every iteration and halt the simulation upon detection.

Importantly, the appearance of NaNs is associated with CFL violation. The advective CFL number remains small and nearly constant throughout the first 2days then it accelerates.
At present, the precise origin of the NaNs remains unresolved. Possible contributing factors include

- unwanted interactions between immersed boundary geometry and the open boundary schemes

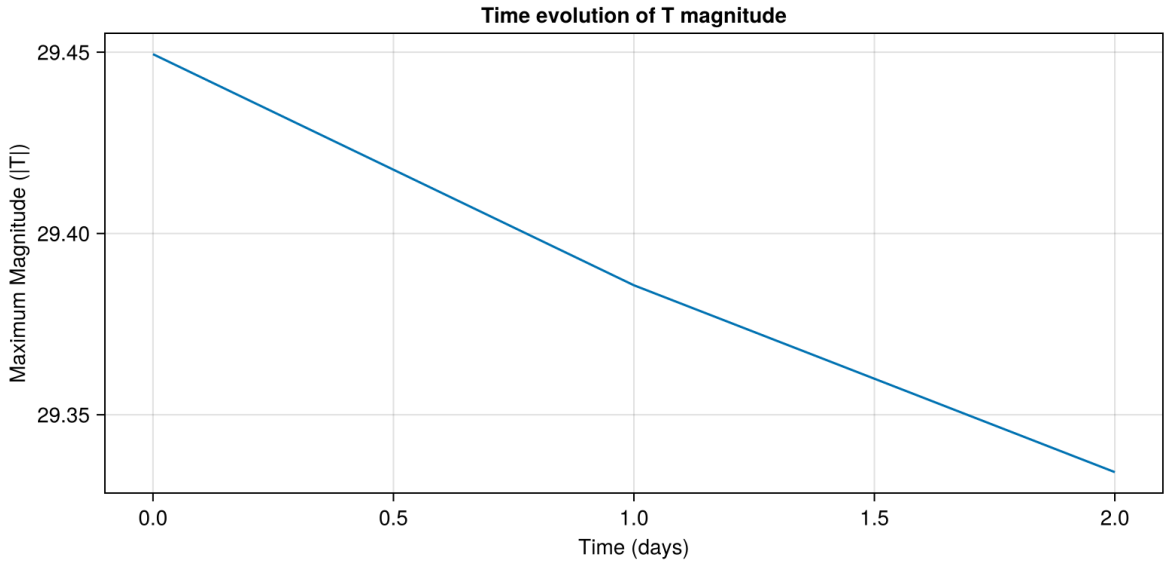

Figure 4.3: Time evolution of the maximum absolute temperature magnitude $|T|_{\max}$ in the two–dimensional Adriatic simulation. The gradual trend indicates stable tracer evolution dominated by advection and boundary forcing.

- inpainting of the GLORYS forcing fields (reconstruct missing, masked, or undefined values along the sub-basin open boundary)
- barotropic solver convergence
- incomplete handling of bidimentional fields and halo regions within the current Oceananigans library.

Given that the Adriatic configuration exercises parts of the Oceananigans codebase that are still under active development, the instability might not necessarily be attributed solely to the perturbation advection boundary condition introduced in this work.

Practical next-step diagnosis. To turn the NaN event into an actionable failure mode, the most useful information is the first location and the first kernel that produces a non-finite value. A minimal protocol can be:

1. Record the first index where a NaN appears (field name, (i, j, k), and boundary side), and save a small neighborhood stencil for all prognostic fields and free-surface variables at the preceding time step.
2. Separate barotropic and baroclinic contributions by checking: (i) the barotropic solver residual history and iteration count at the failing step, (ii) the post-correction boundary mass flux imbalance, and (iii) whether the NaN appears before or after the mass-conservation correction call.
3. Repeat the failing step with boundary updates disabled one-by-one (radiation only, then nudging only, then external clamping) to isolate whether the failure depends on halo writes, immersed-boundary geometry, or external-data inpainting.

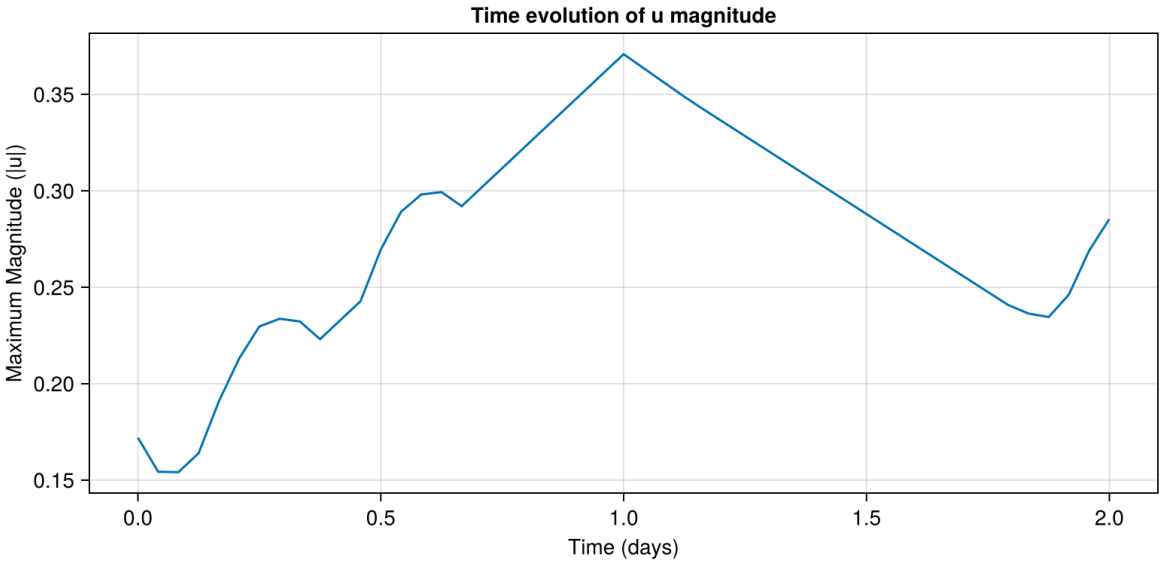

Figure 4.4: Time evolution of the maximum absolute zonal velocity $|u|_{\max}$. The velocity remains bounded and does not exhibit signatures of CFL-driven instability.

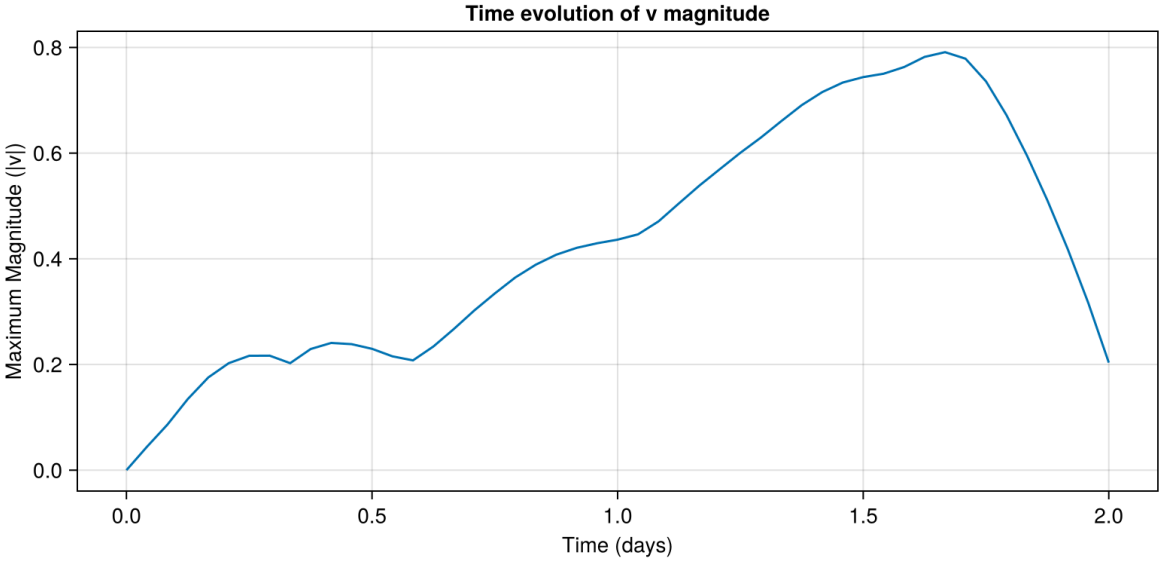

Figure 4.5: Time evolution of the maximum absolute meridional velocity $|v|_{\max}$. Growth reflects large-scale circulation and boundary forcing rather than numerical divergence.

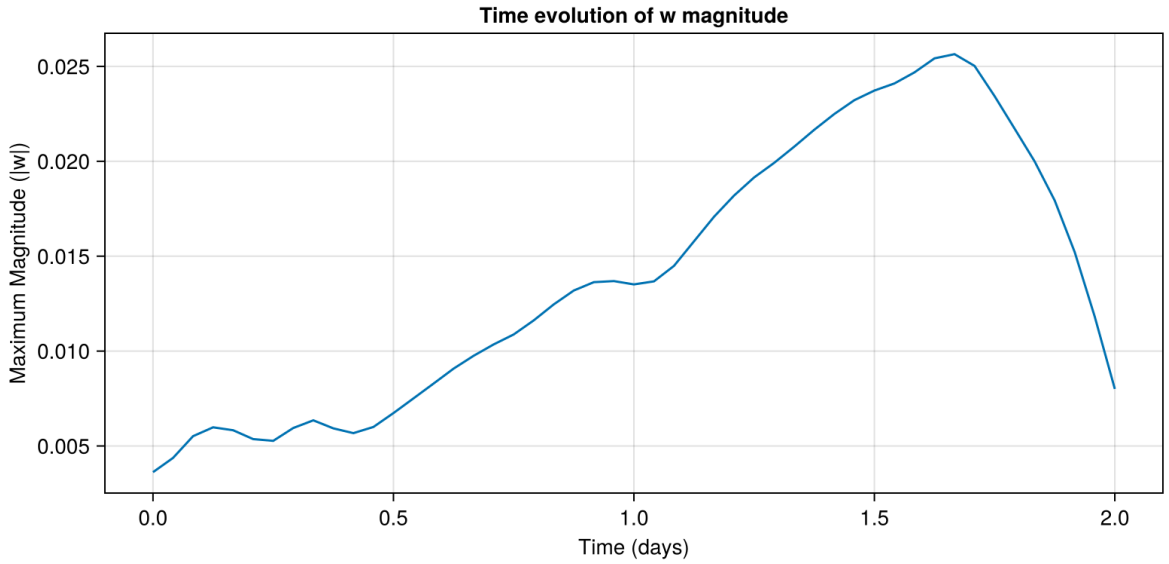

Figure 4.6: Time evolution of the maximum absolute vertical velocity $|w|_{\max}$. Magnitudes remain small relative to horizontal velocities, consistent with hydrostatic dynamics.

4. Add an invariant check after each major update (advection, diffusion, pressure correction, boundary fill). For each field q, verify $\max|q| < q_{\max}$ (a generous physical bound) and $\sum q^2$ does not jump by orders of magnitude in one step without a corresponding forcing term.

This workflow does not assume that the boundary scheme is the sole cause. It allows the failure to be attributed to a specific stage in the time step and a specific data structure (for example, a reduced field halo, an immersed-boundary mask interaction, or an inpainting extrapolation).

Nevertheless, further investigation is required to isolate the source of non–finite values and to assess long–term stability once the remaining infrastructure issues are resolved.

Chapter 5

GPU-portability testing

This chapter documents solver execution measurements obtained with a reduced Oceananigans hydrostatic free-surface configuration. The purpose is to assess GPU portability and execution behavior on the Leonardo supercomputer (CINECA), using a single NVIDIA A100 GPU.

We measure the wall-clock cost of advancing the model state through a fixed number of time steps $N = 10$ using `BenchmarkTools.jl`, not as a benchmarking study, but as a diagnostic tool to characterize execution on the target GPU architecture.

5.1 `BenchmarkTools.jl`

`BenchmarkTools.jl` is the standard Julia package for performance timing. It runs a target expression repeatedly and reports a distribution of runtimes, together with memory and allocation estimates. These statistics help distinguish steady-state throughput from noise due to compilation, scheduling, and runtime variability.

Two macros cover the main use cases:

- `@btime expr` for a quick summary.
- `@benchmark expr` for a full `Trial` object containing all samples.

A minimal example is:

```
using BenchmarkTools
f(x) = sum(abs2, x)
x = rand(Float32, 10^6)
@btime f($x)
```

The `$x` interpolation ensures that `x` is treated as a constant inside the benchmark harness, so the measurement reflects execution time rather than global-variable lookup.

5.2 Execution protocol for GPU portability assessment

The full Adriatic workflow we use for the simulation includes external datasets, inpainting, coupled forcing, and output writers. To isolate GPU execution behavior, we minimize the configuration so that the timed region reflects the core solver cost stepping without external overheads.

The execution retains:

- the hydrostatic free-surface time stepping,
- representative advection schemes used in the development workflow,
- seawater buoyancy and tracer evolution.

The execution avoids:

- file I/O inside the timed region,
- external dataset refresh,
- diagnostic output writers.

Longer execution block for stable timing. Rather than timing a single call to `time_step!`, each execution evaluation advances the model for N consecutive steps. Let t_{tot} be the measured wall time of the full block. The per-step time is

$$t_{\mathrm{step}} = \frac{t_{\mathrm{tot}}}{N}.$$

This approach reduces sensitivity to single-step noise and yields a more stable estimate of execution time per step.

On GPU, kernel launches are asynchronous. The execution wraps the timed block in a GPU synchronization call so that t_{tot} includes kernel completion time, not only kernel launch overhead.

5.2.1 Execution metric for GPU analysis

Throughput metric: Simulated Years Per Day (SYPD). We report an important metric for GPU execution in ocean modeling: throughput using simulated-years-per-wall-day (SYPD), computed from a representative per-step wall time t (seconds) and the simulated time step Δt_{sim} (seconds). In this execution, $\Delta t_{\mathrm{sim}} = 60$ s.
Steps per wall day:

$$N_{\mathrm{steps/day}} = \frac{86400}{t}.$$

Simulated seconds per wall day:

$$t_{\mathrm{sim/day}} = \frac{86400}{t}\,\Delta t_{\mathrm{sim}}.$$

Simulated years per wall day:

$$\mathrm{SYPD} = \frac{t_{\mathrm{sim/day}}}{365 \cdot 86400} = \frac{\Delta t_{\mathrm{sim}}}{365\,t}.$$

This expression shows the direct relationship between per-step wall time and model throughput: reducing t increases SYPD in inverse proportion.

GPU utilization metrics. In addition to throughput, we report hardware-level metrics obtained from GPU monitoring tools (`nvidia-smi`). These metrics provide information on how efficiently the numerical workload utilizes the accelerator.

GPU utilization. The quantity `utilization.gpu` measures the fraction of time during which the GPU is actively executing kernels. Formally, it is a time-averaged occupancy indicator, $U_{\mathrm{GPU}} \in [0, 100]\%$. High values indicate that the computational workload saturates the device, whereas low values indicate under-utilization, typically caused by small problem sizes, kernel launch overhead, or memory-latency-dominated execution. This metric is used to interpret how effectively the workload engages the GPU resources under the tested configuration: idealy, an increase in resolution should lead to higher utilization if the workload approaches the optimal regime.

Memory usage. The quantity `memory.used` reports the allocated device memory, M (MiB). This metric scales approximately with the number of grid cells. Monitoring memory usage allows one to:

- verify consistency with the theoretical memory footprint $\propto N_x N_y N_z$
- detect memory-bound regimes,
- ensure that simulations remain within hardware limits.

Power consumption. The quantity `power.draw` measures the instantaneous power usage of the GPU, P (Watt). This metric is a proxy for the operational regime of the device. Higher power draw typically correlates with higher computational load. In the context of scaling studies, an increase in resolution should lead to an increase in power consumption until the device reaches its nominal operating envelope.

Interpretation for scaling analysis. These metrics complement SYPD by providing a physical interpretation of performance. For the tested configurations, effective GPU utilization is characterized by:

$$U_{\mathrm{GPU}} \uparrow, \quad M \propto N_x N_y N_z, \quad P \uparrow,$$

while maintaining approximately constant

$$\frac{t_{\mathrm{step}}}{N_x N_y N_z}$$

Conversely, low utilization combined with small memory usage indicates that the problem size is insufficient to exploit the available hardware, and throughput measurements in this regime should be interpreted as lower bounds rather than peak performance.

5.2.2 GPU execution model in Julia and Oceananigans

Oceananigans runs on CPUs and GPUs by making the computational kernels depend on an *architecture* object. The same high-level model code constructs fields and tendencies, but the arrays and the generated kernels specialize to the chosen backend.

Step 1: Architecture selects array types. On CPU backends, prognostic fields store data in standard Julia arrays. On NVIDIA backends, fields store data in `CuArray` buffers provided by CUDA.jl. The choice happens when the model is built (for example, by selecting a CUDA architecture), so all subsequent operations inherit the device placement of the underlying arrays.

Step 2: Julia specializes kernels to concrete types. Julia compiles methods for the exact types seen at runtime. Once the model uses `CuArray` fields and GPU architecture objects, Oceananigans launches GPU kernels whose code is specialized to: grid type, halo sizes, numerical scheme types, and array element types. This is why a warmup step is required before timing. The first call triggers compilation, while later calls execute already-compiled kernels.

Step 3: Kernel launches are asynchronous. A GPU kernel launch returns control to the CPU before the kernel finishes. For correct timing, the execution must synchronize the device around the timed region. On NVIDIA GPUs, this means wrapping the execution stepping block inside a `CUDA.@sync` so the measured wall time includes kernel completion, not only launch overhead.

Step 4: Porting pressure points. GPU performance depends strongly on avoiding unnecessary allocations in the stepping loop. Even when arithmetic runs on the GPU, host-side allocations can trigger garbage collection that adds timing variability. The allocation counts reported by BenchmarkTools therefore act as a direct signal of where the code still builds temporary objects during stepping. The practical remediation is to preallocate buffers, avoid temporary array construction in callbacks, and keep per-step logic type-stable so Julia can generate a single efficient kernel path.

Step 5: What "write once" means here. The boundary-condition algorithms and solver modifications in this thesis remain written in the same Julia source form for CPU and GPU execution. The performance portability comes from (1) array placement controlled by the architecture choice, and (2) compilation of backend-specific kernels from the same method definitions.

5.3 Leonardo (CINECA) execution environment

5.3.1 GPU run:

For the GPU environment check table 5.1

GPU hardware We verified the device and driver stack with `nvidia-smi`. The GPU used for execution is:

- NVIDIA A100-SXM with 64 GB memory,
- NVIDIA driver version 535.274.02,
- CUDA version 12.2.

For traceability, the `nvidia-smi` header captured during the run is:

```
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.274.02             Driver Version: 535.274.02   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
```

```
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA A100-SXM-64GB           On  | 00000000:C8:00.0 Off |                    0 |
| N/A   44C    P0              61W / 457W |      0MiB / 65536MiB |      0%      Default |
|                                         |                      |             Disabled |
+-----------------------------------------+----------------------+----------------------+
```

Component	Leonardo execution configuration
GPU	NVIDIA A100-SXM-64GB
GPU memory	64 GB
Driver	535.274.02
CUDA	12.2
Allocation	1 node, 1 task, 1 GPU

Table 5.1: Leonardo (CINECA) GPU configuration used for execution measurements.

5.3.2 CPU run:

For the CPU environment check table 5.2

CPU hardware We verified the CPU configuration using `lscpu`. The CPU used for execution is:

- Intel(R) Xeon(R) Platinum 8480+,
- 2 sockets, 56 cores per socket (112 cores total),
- 1 thread per core,
- base frequency 2001 MHz,
- L3 cache 107520 KB.

For traceability, a condensed `lscpu` header captured during the run is:

```
Architecture:        x86_64
CPU(s):              112
Thread(s) per core:  1
Core(s) per socket:  56
Socket(s):           2
Vendor ID:           GenuineIntel
Model name:          Intel(R) Xeon(R) Platinum 8480+
CPU MHz:             2001.000
L3 cache:            107520K
NUMA node(s):        8
```

Component	Leonardo execution configuration
CPU	Intel Xeon Platinum 8480+
Sockets	2
Cores	112 total (56 per socket)
Threads/core	1
Base frequency	2001 MHz
NUMA domains	8
Allocation	1 node, 1 task, 1 CPU core

Table 5.2: Leonardo (CINECA) CPU configuration used for execution measurements.

Warmup and device synchronization. Julia compiles kernels on first call, and Oceananigans triggers kernel specialization during initial stepping. The execution performs one warmup step before timing.

On accelerators, asynchronous kernel launches require an explicit synchronization around the timed region. On Metal backends this uses `Metal.@sync`. On NVIDIA backends the analogous pattern uses `CUDA.@sync`. Synchronization ensures the execution measures full kernel completion time rather than launch latency.

5.4 GPU execution results on Leonardo

This section reports performance measurements obtained with the reduced hydrostatic Oceananigans execution described in Section 5.2. The execution advances the model for blocks of $N = 100$ time steps and reports throughput in simulated years per wall day (SYPD), with $\Delta t_{\text{sim}} = 60\,\text{s}$. The Leonardo execution environment consists of one Intel Xeon Platinum 8480+ CPU core for the CPU run and one NVIDIA A100-SXM-64GB GPU for the accelerator run.

5.4.1 One CPU core versus one GPU for a fixed grid

We first compare CPU and GPU performance for the same execution configuration on the grid

$$(N_x, N_y, N_z) = (150, 50, 30).$$

For this fixed problem size, the minimum per-step wall time on one CPU core is

$$t_{\text{step}}^{\text{CPU}} = 9.0096 \times 10^{-2}\ \text{s},$$

which corresponds to

$$\text{SYPD}^{\text{CPU}} = 1.8245.$$

On one NVIDIA A100 GPU, the minimum per-step wall time is

$$t_{\text{step}}^{\text{GPU}} = 1.4137 \times 10^{-3}\ \text{s},$$

which corresponds to

$$\text{SYPD}^{\text{GPU}} = 116.283.$$

The resulting speedup for this execution is therefore

$$\frac{t_{\text{step}}^{\text{CPU}}}{t_{\text{step}}^{\text{GPU}}} = \frac{\text{SYPD}^{\text{GPU}}}{\text{SYPD}^{\text{CPU}}} \approx 63.73.$$

This result shows that, for the reduced hydrostatic execution considered here, a single A100 GPU delivers about two orders of magnitude more throughput than a single CPU core. In practical terms, the same Julia/Oceananigans model that advances at about 1.82 simulated years per wall day on one CPU core reaches about 116 simulated years per wall day on one GPU. This confirms that the Oceananigans implementation used in this thesis benefits strongly from GPU execution on Leonardo.

The memory and allocation statistics also differ qualitatively between the two runs. The CPU case exhibits a very small garbage-collection fraction, so its runtime is dominated by numerical work. By contrast, the GPU case still shows a non-negligible garbage-collection fraction together with a substantially larger allocation count. This indicates that, although the numerical throughput on GPU is already high, the stepping loop still performs host-side allocations that contribute to runtime variability. Consequently, the measured 63.7× speedup should be interpreted as the performance of the current implementation, not as an upper bound on what the same execution could achieve after allocation reduction.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This thesis addressed two concrete goals in the context of regional ocean modeling with modern HPC tools.

Open boundary algorithm development. We implemented a perturbation-advection open boundary condition for the hydrostatic free-surface solver in `Oceananigans.jl`. In one dimension, the boundary diagnoses an outward phase speed from interior history, bounds it by a CFL-safe limit, and updates the boundary with an implicit stencil. Idealized barotropic tests show clean radiation of the free-surface perturbation, with domain-wide maxima of $|\eta|$ and $|u|$ decaying to near zero over multi-day runs. This supports the claim that the 1D formulation behaves as intended for purely outward-propagating disturbances.

We then extended the method to two dimensions using a Raymond–Kuo style generalization that includes tangential gradients and optional direction-dependent nudging. In a realistic North Adriatic Sea configuration forced by GLORYS and coupled to atmospheric fluxes, the model remains stable for short integrations with small advective CFL. The integration still terminates at longer times due to NaN emergence, which indicates that robust long-term regional configurations require additional improvements in the surrounding infrastructure (halo handling in non-simple geometries, immersed-boundary interactions, inpainting robustness, and barotropic solver safeguards).

Benchmarking, GPU acceleration, and single-GPU problem-size scaling. We benchmarked a reduced hydrostatic free-surface configuration on the Leonardo (CINECA) system under a controlled protocol using the simulated-years-per-wall-day (SYPD) metric. For the fixed benchmark grid $(150, 50, 30)$, a single NVIDIA A100 GPU achieves approximately 116 simulated years per wall day, compared with approximately 1.82 on a single CPU core, corresponding to a speedup of about $64\times$. This confirms that the Julia plus Oceananigans approach can deliver substantial accelerator throughput without rewriting the scientific model in a separate GPU language.

We then extended the benchmark to a single-GPU problem-size study by increasing the grid resolution while keeping the solver configuration unchanged. The results do not indicate ideal linear scaling over the full tested range. Instead, they show three regimes: an under-filled small-problem regime, a more efficient intermediate regime, and a degradation regime for larger grids. In particular, the smallest case under-utilizes the GPU, the intermediate grid $(300, 100, 30)$ gives the best normalized performance among the tested

cases, and larger grids exhibit increasing cost per cell together with reduced throughput. Therefore, the current implementation already provides strong GPU acceleration, but the scaling results also indicate that further performance engineering—especially reduction of allocations and analysis of memory-traffic and kernel-efficiency limits—is still required to approach the full potential of the hardware.

6.2 Future Work

6.2.1 Boundary-condition algorithm extensions

- Implement a full radiation-nudging scheme for tracers and momentum that mirrors the ROMS RadNud logic, including a robust inflow detector that combines phase-speed diagnostics with the sign of normal transport.
- Add barotropic-specific boundary options (Chapman for η, Flather for depth-integrated velocity) and couple them consistently to the hydrostatic free-surface solver so that long-time mass and sea-level drift remain controlled.
- Revisit the 2D diagnostic to reduce sensitivity when interior gradients are small. This can be done by adding slope limiters or by blending toward external data whenever the diagnostic becomes ill-conditioned.

6.2.2 Stability of realistic regional configurations

- Isolate the NaN source by recording the first failing index, the first failing update stage, and the local stencil of prognostic variables. Use controlled toggles (boundary update off, inpainting off, immersed boundary off) to identify the minimal failing configuration.
- Strengthen halo and reduced-field boundary handling, with explicit tests for 2D reduced fields and immersed-boundary masks. Add regression tests that reproduce the North Adriatic Sea simulation failure in a small domain.
- Improve external-data inpainting and interpolation near steep bathymetry and coastlines, since small non-physical gradients near masked cells can inject large tendencies into advection schemes.

6.2.3 Performance engineering and scaling

- Reduce allocations in the stepping loop by preallocating scratch buffers and ensuring callback logic does not construct temporary arrays. This should reduce GC overhead and tighten benchmark variability, especially on GPU.
- Extend benchmarking from single-device throughput to strong and weak scaling, including MPI runs and multi-GPU configurations. Use the same SYPD metric to report scaling in a form that is standard in ocean modeling performance work.
- Upstream the general solver fixes (reduced fields at boundaries, mass-flux tracking for the hydrostatic model, unit-consistent coupling) to reduce divergence from

Oceananigans mainline and make the workflow reproducible for other regional domains.

Bibliography

[1] S. Silvestri, G. L. Wagner, C. Hill, M. R. Ardakani, J. Blaschke, J.-M. Campin, V. Churavy, N. C. Constantinou, A. Edelman, J. Marshall, *et al.*, "Oceananigans. jl: A julia library that achieves breakthrough resolution, memory and energy efficiency in global ocean simulations," *arXiv preprint arXiv:2309.06662*, 2023.

[2] G. L. Wagner, S. Silvestri, N. C. Constantinou, A. Ramadhan, J.-M. Campin, C. Hill, T. Chor, J. Strong-Wright, X. K. Lee, F. Poulin, *et al.*, "High-level, high-resolution ocean modeling at all scales with oceananigans," *arXiv preprint arXiv:2502.14148*, 2025.

[3] L. P. Røed and C. K. Cooper, *Open Boundary Conditions in Numerical Ocean Models*, pp. 411–436. Dordrecht: Springer Netherlands, 1986.

[4] I. Orlanski, "A simple boundary condition for unbounded hyperbolic flows," *Journal of Computational Physics*, vol. 21, no. 3, pp. 251–269, 1976.

[5] A. F. Shchepetkin and J. C. McWilliams, "The regional oceanic modeling system (roms): a split-explicit, free-surface, topography-following-coordinate oceanic model," *Ocean Modelling*, vol. 9, no. 4, pp. 347–404, 2005.

[6] G. Madec, M. Bell, R. Benshila, A. Blaker, R. Boudrallé-Badie, C. Bricaud, D. Bruciaferri, D. Carneiro, M. Castrillo, D. Calvert, J. Chanut, E. Clementi, A. Coward, C. de Lavergne, S. Dobricic, I. Epicoco, C. Éthé, E. Fiedler, D. Ford, R. Furner, J. Ganderton, T. Graham, J. Harle, K. Hutchinson, D. Iovino, R. King, D. Lea, C. Levy, T. Lovato, E. Maisonnave, J. Mak, J. M. C. Sanchez, M. Martin, N. Martin, D. Martins, S. Masson, P. Mathiot, F. Mele, S. Mocavero, A. Moulin, S. Müller, G. Nurser, P. Oddo, S. Paronuzzi, J. Paul, M. Peltier, R. Person, C. Rousset, S. Rynders, G. Samson, D. Schroeder, D. Storkey, A. Storto, S. Téchené, M. Vancoppenolle, and C. Wilson, "Nemo ocean engine reference manual," Dec. 2024.

[7] J. Marshall, A. Adcroft, C. Hill, L. Perelman, and C. Heisey, "A finite-volume, incompressible navier stokes model for studies of the ocean on parallel computers," *Journal of Geophysical Research: Oceans*, vol. 102, no. C3, pp. 5753–5766, 1997.

[8] A. Artegiani, R. Azzolini, and E. Salusti, "On the dense water in the adriatic sea," *Oceanologica Acta*, vol. 12, no. 2, pp. 151–160, 1989.

[9] I. Vilibić, P. Pranić, and C. Denamiel, "North adriatic dense water: lessons learned since the pioneering work of mira zore-armanda 60 years ago," *Acta Adriatica*, vol. 64, no. 1, pp. 53–78, 2023.

[10] W. H. Raymond and H. Kuo, "A radiation boundary condition for multi-dimensional flows," *Quarterly Journal of the Royal Meteorological Society*, vol. 110, no. 464, pp. 535–551, 1984.

[11] P. Marchesiello, J. C. McWilliams, and A. Shchepetkin, "Open boundary conditions for long-term integration of regional oceanic models," *Ocean modelling*, vol. 3, no. 1-2, pp. 1–20, 2001.

[12] D. C. Chapman, "Numerical treatment of cross-shelf open boundaries in a barotropic coastal ocean model," *Journal of Physical Oceanography*, vol. 15, no. 8, pp. 1060 – 1075, 1985.

[13] E. D. Palma and R. P. Matano, "On the implementation of passive open boundary conditions for a general circulation model: The barotropic mode," *Journal of Geophysical Research: Oceans*, vol. 103, no. C1, pp. 1319–1341, 1998.

[14] S. Silvestri, G. L. Wagner, N. C. Constantinou, C. N. Hill, J.-M. Campin, A. N. Souza, S. Bishnu, V. Churavy, J. Marshall, and R. Ferrari, "A gpu-based ocean dynamical core for routine mesoscale-resolving climate simulations," *Journal of Advances in Modeling Earth Systems*, vol. 17, no. 4, p. e2024MS004465, 2025. e2024MS004465 2024MS004465.

Appendices

Appendix A

Boundary update asymptotics

A.1 Why the boundary update can ignore the source term at leading order.

Near an open boundary, we want a local update rule that transports interior perturbations outward. Write the boundary-normal velocity as $u = U + u'$, where U is a slowly varying background and u' carries the fast boundary-relevant disturbance. If we linearize the momentum tendency around U, we obtain the schematic form

$$\partial_t u' + U\,\partial_x u' = S, \tag{A.1}$$

where the residual S collects terms that do not behave like pure advection of u' (for example, tendencies from slow variations in U, pressure corrections applied elsewhere in the split free-surface update, or other forcing contributions).

To justify a radiation-type update, we compare the size of the source-driven increment to the advection-driven increment over one step:

$$\Delta u'_{\mathrm{src}} \sim S\,\Delta t, \qquad \Delta u'_{\mathrm{adv}} \sim U\,(\partial_x u')\,\Delta t.$$

This motivates the dimensionless ratio

$$\epsilon := \frac{\Delta u'_{\mathrm{src}}}{\Delta u'_{\mathrm{adv}}} \sim \frac{S}{U\,\partial_x u'}. \tag{A.2}$$

When $\epsilon \ll 1$, advection dominates the boundary evolution of u', so a Sommerfeld-style diagnosis based on $\partial_t u'$ and $\partial_x u'$ remains a good approximation for the local outgoing mode. In practice, we enforce this numerically by zeroing the diagnostic when the interior gradient becomes too small (to avoid division by noise), and by bounding the diagnosed speed by a CFL-safe upper limit.

$$\tilde{U} = -\frac{u^{\mathrm{new}}_{B-1} - u^{\mathrm{old}}_{B-1}}{u^{\mathrm{new}}_{B-1} - u^{\mathrm{new}}_{B-2}} \tag{A.3}$$

where B stands for boundary point.

A.2 Derivation of equation (3.2)

Near an open boundary we do not attempt to retain the full interior dynamics. Instead, we isolate the part of the dynamics that is most relevant for boundary transparency,

namely the outward transport of disturbances relative to a slowly varying background flow.

Starting from the momentum equation

$$\partial_t u + (u \cdot \nabla)u = -\nabla P + F, \tag{A.4}$$

we decompose the velocity into a prescribed background state and a perturbation, $u = U + u'$. Substituting this decomposition gives

$$\partial_t(U + u') + (U + u') \cdot \nabla(U + u') = -\nabla P + F. \tag{A.5}$$

We now expand each term carefully. Since $\partial_t(U + u') = \partial_t U + \partial_t u'$, and

$$(U + u') \cdot \nabla(U + u') = U \cdot \nabla U + U \cdot \nabla u' + u' \cdot \nabla U + u' \cdot \nabla u',$$

we obtain

$$\partial_t U + \partial_t u' + U \cdot \nabla U + U \cdot \nabla u' + u' \cdot \nabla U + u' \cdot \nabla u' = -\nabla P + F. \tag{A.6}$$

At this stage the equation is still exact. The boundary model is obtained by a sequence of approximations valid only in a thin neighborhood of the open boundary.

First, the background field is assumed to vary slowly in time and space relative to the boundary perturbation. This means that

$$\partial_t U \approx 0, \qquad U \cdot \nabla U \approx 0. \tag{A.7}$$

Second, the perturbation is assumed sufficiently small that quadratic self-interaction can be neglected at leading order:

$$u' \cdot \nabla u' \approx 0. \tag{A.8}$$

Third, the background is taken approximately uniform near the boundary, so that

$$\nabla U \approx 0 \qquad \Longrightarrow \qquad u' \cdot \nabla U \approx 0. \tag{A.9}$$

Finally, in the original `PerturbationAdvection` construction, the pressure-gradient contribution is not treated as part of the local boundary radiation step, and is therefore neglected in this reduced boundary model:

$$\nabla P \approx 0. \tag{A.10}$$

After applying these assumptions, the exact decomposed momentum equation reduces to

$$\partial_t u' + U \cdot \nabla u' \approx F. \tag{A.11}$$

The forcing is then modeled as a linear relaxation of the perturbation toward the external state,

$$F = -\frac{u'}{\tau}, \tag{A.12}$$

where τ is the nudging timescale. Hence the perturbation satisfies the local advection-relaxation equation

$$\partial_t u' + U \cdot \nabla u' = -\frac{u'}{\tau}. \tag{A.13}$$

We now express this reduced equation in terms of the full velocity u. Since $u = U + u'$ then we have $u' = u - U$. Differentiating in time gives $\partial_t u' = \partial_t u - \partial_t U$. Using again the slow-background assumption $\partial_t U \approx 0$, we obtain

$$\partial_t u' \approx \partial_t u. \tag{A.14}$$

Substituting this into (A.13) yields

$$\partial_t u + U \cdot \nabla u' \approx -\frac{u'}{\tau}, \tag{A.15}$$

and therefore

$$\partial_t u \approx -U \cdot \nabla u' - \frac{u'}{\tau}. \tag{A.16}$$

Equation (A.16) is the boundary evolution law used by the `PerturbationAdvection` scheme. It should be interpreted carefully: it is not the full momentum equation rewritten in another form, but a reduced local model obtained by retaining only the two mechanisms that the boundary condition is meant to represent, namely advection of the perturbation by the background flow and relaxation of that perturbation toward the prescribed exterior state.

A.3 Orlanski logic for exiting the domain

The purpose of this section is to verify that the discrete Orlanski boundary update admits a purely outgoing wave without generating reflection. If the boundary condition is transparent, an outgoing Fourier mode should satisfy the boundary equation exactly.

Consider therefore a single wave component written in discrete space and time as

$$u_J^n = u_0 \exp\Big(i(kJ\Delta x - \omega n \Delta t)\Big). \tag{A.17}$$

It is convenient to introduce the shorthand

$$S = e^{ik\Delta x}, \qquad R = e^{-i\omega\Delta t}, \tag{A.18}$$

so that the wave can be written compactly as

$$u_J^n = u_0 S^J R^n. \tag{A.19}$$

The discrete form of the radiation update used at the boundary point J is

$$u_J^{n+1} = u_J^n - \frac{\Delta t}{\Delta x} c \left(u_J^{n+1} - u_{J-1}^{n+1}\right), \tag{A.20}$$

where the phase speed c is diagnosed from interior values according to

$$c = -\frac{\Delta x}{\Delta t} \frac{u_{J-1}^{n+1} - u_{J-1}^n}{u_{J-1}^{n+1} - u_{J-2}^{n+1}}. \tag{A.21}$$

Substituting the Fourier mode $u_J^n = u_0 S^J R^n$ into the diagnostic formula gives

$$u_{J-1}^{n+1} - u_{J-1}^n = u_0 S^{J-1}(R^{n+1} - R^n) = u_0 S^{J-1} R^n (R - 1),$$

and

$$u_{J-1}^{n+1} - u_{J-2}^{n+1} = u_0 R^{n+1}(S^{J-1} - S^{J-2}) = u_0 R^{n+1} S^{J-2}(S-1).$$

After cancelling common factors one obtains

$$c = \frac{\Delta x}{\Delta t} \frac{S(R-1)}{R(S-1)}. \tag{A.22}$$

Using the Fourier representation S^J and R^n the boundary equation becomes

$$S^J R^{n+1} = S^J R^n - \frac{\Delta t}{\Delta x} c \left(S^J R^{n+1} - S^{J-1} R^{n+1} \right).$$

Factoring the common terms gives

$$S^J R^{n+1} = S^J R^n - \frac{\Delta t}{\Delta x} c S^{J-1} R^{n+1}(S-1).$$

Dividing both sides by $S^J R^{n+1}$ yields

$$1 = R^{-1} - \frac{\Delta t}{\Delta x} c \frac{S-1}{S}. \tag{A.23}$$

Substituting the diagnosed phase speed gives

$$1 = R^{-1} - \frac{S(R-1)}{R(S-1)} \frac{S-1}{S}.$$

After cancelling the common factors the expression reduces to

$$1 = R^{-1} + \frac{R-1}{R}.$$

The identity holds exactly, which means that the outgoing Fourier mode satisfies the discrete boundary equation without the need for an additional reflected component. The Orlanski phase-speed diagnostic therefore removes the mismatch between the temporal and spatial discrete derivatives, allowing the wave to propagate through the boundary without reflection.