ISBN 978-82-326-4656-2 (printed version)
ISBN 978-82-326-4657-9 (electronic version)
ISSN 1503-8181

NTNU
Norwegian University of
Science and Technology

Håvard Heitlo Holm

Doctoral Thesis

Doctoral theses at NTNU, 2020:154

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology
and Electrical Engineering
Department of Mathematical Sciences

NTNU

Doctoral theses at NTNU, 2020:154

Håvard Heitlo Holm

Efficient Forecasting of Drift Trajectories using Simplified Ocean Models and Nonlinear Data Assimilation on GPUs

NTNU
Norwegian University of
Science and Technology

Håvard Heitlo Holm

Efficient Forecasting of Drift Trajectories using Simplified Ocean Models and Nonlinear Data Assimilation on GPUs

Thesis for the degree of Philosophiae Doctor

Trondheim, May 2020

Norwegian University of Science and Technology
Faculty of Information Technology
and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

Faculty of Information Technology
and Electrical Engineering
Department of Mathematical Sciences

© Håvard Heitlo Holm

ISBN 978-82-326-4656-2 (printed version)
ISBN 978-82-326-4657-9 (electronic version)
ISSN 1503-8181

Doctoral theses at NTNU, 2020:154

Printed by Skipnes Kommunikasjon as

Abstract

This thesis presents research on efficient, massively parallel methods and algorithms related to short-term forecasting of drift trajectories in the ocean. The topic has clear societal applications and is an important tool for, e.g., search-and-rescue operations at sea, planning of oil-spill cleanup, and early collision detection between icebergs and offshore installations.

In this work, we investigate computational techniques that can be used complementary to the operational methods already in place today. The traditional approach is to use complex ocean models, of which it is only feasible to run a small ensemble. Due to large uncertainties in initial conditions for oceanographic simulations, however, we propose to use simplified ocean models that capture the relevant physics on short time horizons. We base our simplified ocean models on the rotational shallow-water equations, simulated using an explicit, high-resolution, finite-volume scheme. Since such schemes can be implemented to run efficiently on the graphics processing unit (GPU), we can afford to run a large ensemble of simplified ocean models. Furthermore, we investigate nonlinear data-assimilation techniques, such as particle filters, that enable us to use available observations of the ocean state to reduce the uncertainty in the ensemble. Our hope is that this approach, possibly in combination with the operational methods, can give a more complete picture of the uncertainties in the forecasted drift trajectories.

The thesis consists of an introductory part plus five scientific papers. The first two papers assess enabling technologies and methods needed for our approach to forecasting of drift trajectories. This includes evaluating numerical schemes for their suitability to capture oceanographic shallow-water flow, and assessing programming environments for GPU computing. The third paper presents a massively parallel algorithm for applying the recently proposed implicit equal-weights particle filter to a shallow-water model for forecasting of drift trajectories. In the fourth paper, we present a framework for running efficient oceanographic simulations using a modern finite-volume scheme initiated from operational ocean circulation forecasts. Finally, the fifth paper explores the possibility of using a very large ensemble with 10 000 members along with a basic particle filter method for ensemble prediction of drift trajectories.

Contents

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of philosophiae doctor (PhD) at the Norwegian University of Science and Technology (NTNU), Trondheim, Norway.

The work has been performed at the Department of Mathematics and Cybernetics, SINTEF Digital, Oslo, and the Department of Mathematical Sciences, NTNU, Trondheim. It was financed by the Norwegian Research Council through the GPU Ocean project, grant number 250935.

Acknowledgments

This thesis has not been written in a vacuum and many deserve my greatest gratitude for helping me get through. I entered this project with reasonable knowledge on numerical mathematics and GPU computing, and now that the project is finished I have gained a deeper insight in those fields while also expanded my repertoire to include computational statistics, data assimilation, and oceanography.

First, I would like to thank my supervisors for their effort and guidance: Dr. André Rigland Brodtkorb, for always bringing new ideas and extra enthusiasm to the table; Dr. Martin Lilleeng Sætra, for his thoroughness and for exploring all the restaurants in Reading with me; and Prof. Knut-Andreas Lie, for sharing his insights on the bigger picture. Thanks also to the rest of the GPU Ocean team, and to all who worked years and years to get this project funded. I am grateful that I was offered the opportunity to work in this project and it has been inspiring to work with so many talented people. Thanks to Prof. Peter Jan van Leeuwen for hosting me and Martin at the Data Assimilation Research Centre at the University of Reading, UK, for three months during the winter of 2018, and for guiding us into the world of data assimilation.

I also want to thank all my colleagues in the Computational Geosciences group and the Visualisation and Accelerated Computing group here at SINTEF, for fruitful discussions, continuous supplies of bad jokes, and deep exploration of the Norwegian language around the coffee machine.

Furthermore, my deepest thanks goes to my ever supporting family, and especially to my parents for giving me such a solid foundation. Thanks also to my friends here in Oslo for not giving up on inviting me to go climbing even though this thesis have limited my spare time lately, and to my friends back home in Verdal'n for keeping our Second Christmas Day traditions alive and well.

And finally, of course, my biggest thanks goes to Oda. Now, as it is your turn to take on your professional quest, I will do my uttermost to be as supportive for you as you have been for me.

Håvard Heitlo Holm
Oslo, February 2020

List of Papers

Papers included in thesis

I: **Evaluation of Selected Finite-Difference and Finite-Volume Approaches to Rotational Shallow-Water Flow**
Håvard Heitlo Holm, André Rigland Brodtkorb, Göran Broström, Kai H. Christensen, Martin Lilleeng Sætra
Communications in Computational Physics, volume 27, number 4, pages 1234–1274, 2020
DOI: 10.4208/cicp.OA-2019-0033

II: **GPU Computing with Python: Performance, Energy Efficiency and Usability**
Håvard Heitlo Holm, André Rigland Brodtkorb, Martin Lilleeng Sætra
Computation, volume 8, number 1:4, 2020
(Special issue on Energy-Efficient Computing on Parallel Architectures)
DOI: 10.3390/computation8010004

III: **Massively Parallel Implicit-Equal Weights Particle Filter for Ocean Drift Trajectory Forecasting**
Håvard Heitlo Holm, Martin Lilleeng Sætra, Peter Jan van Leeuwen
Journal of Computational Physics: X, volume 6, number 100053, 2020.
DOI: 10.1016/j.jcpx.2020.100053

IV: **Real-World Oceanographic Simulations on the GPU using a Two-Dimensional Finite Volume Scheme**
André Rigland Brodtkorb, Håvard Heitlo Holm
Preprint: arXiv:1912.02457, 2019
In review

V: **Data Assimilation for Ocean Drift Trajectories Using Massive Ensembles and GPUs**
Håvard Heitlo Holm, Martin Lilleeng Sætra, André Rigland Brodtkorb
Accepted for publication in the proceedings of the conference
Finite Volumes for Complex Applications IX, 2020

Papers not included in thesis

The following conference paper was written as part this thesis, but has later been reworked and extended into a journal version in Paper II.

VI: **Performance and Energy Efficiency of CUDA and OpenCL for GPU Computing using Python**
Håvard Heitlo Holm, André Rigland Brodtkorb, Martin Lilleeng Sætra
Advances in Parallel Computing, volume 36, pages 593–604, 2020
DOI: 10.3233/APC200089

Part I

Background

Chapter 1

Introduction

This thesis explores topics related to forecasting drift trajectories in the ocean. It is a continuation of a research activity at the Department of Mathematics and Cybernetics at SINTEF Digital, focusing on applying modern high-resolution schemes to various applications of shallow-water flows, an activity that has been ongoing for the last 15 years (see, e.g., [61, 8, 82]). The research conducted herein therefore builds on solid experience of developing such numerical methods and researching efficient algorithms for solving them on graphics processing units (GPUs). With this work, we extend this activity into the ocean by expanding our knowledge in oceanography, and diving into the, for us until now, relatively unknown world of data assimilation. This thesis presents three years of research and new experiences in that aspect.

1.1 Background

Precise short-term predictions of drift trajectories in the ocean have several important applications [21]. First of all, they are crucial to limit search regions during search and rescue operations at sea [7]. In the oil sector, increased activity in the Barents Sea makes installations more prone to collisions with icebergs, and forecasts of their trajectories can have a central role in the decision making for evacuating a platform and stop production. Also in case of oil spill, trajectory prediction tools are essential for planning the placement of lenses during oil spill cleaning up [78].

The dynamics of the ocean has many similarities with the atmosphere and consists of currents driven by high and low pressure systems, temperature and salinity gradients, the rotation of the earth, the topography of the sea floor and coast line, and the weather (see [30] or any other textbook on atmosphere-ocean dynamics). By carefully building mathematical models of the dynamical state of the ocean and the influences from these factors, it is possible to predict future states of the ocean in much the same way as we calculate the weather forecast. Besides mathematical modelling and simulation, the third main ingredient in ocean forecasting is data assimilation. This is a mathematical technique used to continuously improve the initial conditions of the forecast (the situation *right now* from which we start the forecast simulation) by incorporating measurements from weather stations and observations made from instruments such as radar and satellites.

Even though ocean forecasting is conceptually very similar to weather forecasting, the problem is much harder to solve for two main reasons. First, there are very few observations of the ocean[1], which makes it much harder to give a precise description of the initial state from which to produce a forecast [35]. Second, the horizontal scales of the features of the ocean are much smaller than their atmospheric relatives. High- and low-pressure systems in the atmosphere typically have a horizontal length scale of 1000 km, whereas similar systems in the ocean are on the scale of 25–50 km. This means that we need to use a higher horizontal resolution for the ocean models compared to what is used in corresponding atmospheric models if we wish to obtain the same level of detail in both. On the other side, the ocean dynamics is slower and thereby plays on a longer time scale compared to the dynamics in the atmosphere.

Whereas the drift trajectory for an object sufficiently deep (> 15 m) beneath the ocean surface can be simulated reasonably well just from the ocean currents, wind and waves also need to be considered for objects at the surface [21]. The wind forcing, for instance, influences both the uppermost ocean layer via Ekman transport and the object directly, whereas surface waves contribute with so-called Stokes drift, caused by the net forward motion of the waves. Furthermore, the shape of the object and the exact depth are important factors that determine the influence of each of these forces, as illustrated by the real-life drifter experiments in Figure 1.1 [81]. Half-submerged spherical drifters are strongly influenced by wind forcing and hence follow very different trajectories compared to fully submerged cross-shaped drifters, for which the wind forcing is negligible. This is also one of the reasons why it is hard to predict the trajectories for icebergs [13].

In Norway, the Norwegian Meteorological Institute (MET Norway) has

[1]According to a lecture I attended on ocean data assimilation at the data assimilation course at ECMWF (the European Centre for Medium-Range Weather Forecasts), the number of observations of the ocean is less than 1% of the number of observations of the atmosphere.

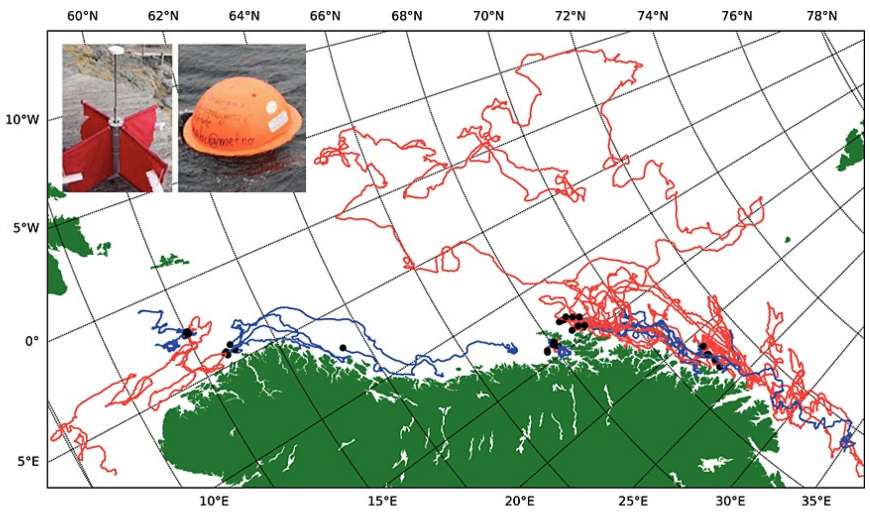


Figure 1.1: Two different drifters and their experimental drift trajectories. The iSphere drifters (to the right with blue trajectories) are spherical and half submerged, whereas the CODE drifters (left and in red) have drouges that are totally submerged. Both are influenced by the currents in the uppermost part of the ocean, but the iSphere drifters also experience a contribution from wind and waves. Figure by Röhrs and Christensen [81], published under the creative commons CC BY-NC-ND 4.0 license.

the national operational responsibility for providing forecasts of ocean currents and drift trajectories. Ocean currents are predicted using the Regional Ocean Modeling System (ROMS) [86], which models the three-dimensional dynamics of the ocean. It is run on a daily basis for three different areas, each with different horizontal resolution, and is very computationally demanding. For instance, the NorShelf model system [79] runs ROMS on an area covering the Norwegian continental shelf using 2.4 km horizontal resolution (900×350 grid points) and 42 vertical layers, while also assimilating all available and relevant observations. Drift trajectories, however, are computed through the light-weight OpenDrift software [23], which reads the resulting ocean forecast along with the relevant weather forecast, and uses these to simulate drift trajectories for objects of different characteristics. Since there are uncertainties in the forecasts and in the initial positions of each object, OpenDrift can run ensembles of trajectories, using slightly perturbed versions of the input forcing fields (currents, winds, waves, etc.). The system is, however, unable to use potential new observations to influence the forcing fields, as it just passively applies the ocean states.

1.2 The aim of this thesis

The aim of this thesis is to investigate an alternative approach to forecasting drift trajectories in applications that essentially are short-term problems and thus require forecasts to be valid for a time span from a few hours and up to a few days. Within this time range, the horizontal currents can be estimated by a simplified ocean model based on the shallow-water equations. These equations can be solved much more efficiently for three reasons. First, they are less complex and therefore typically require fewer mathematical operations to be evolved forward in time. Second, they use a two-dimensional instead of a three-dimensional representation of the ocean state and fewer physical parameters, meaning that there are less data to process. And third and foremost, the shallow-water equations are purely hyperbolic and can therefore be solved using explicit methods, in contrast to the primitive equations which also contain elliptic parts that induce the solution of linear equation systems. This means that the simplified model can be simulated very efficiently by the use of graphics processing units (GPUs) [34, 12]. An additional advantage of the simplified models is that they contain fewer balance relations than the full ocean states, and therefore are easier to perturb. Because of this, we can afford to run an ensemble of such models, each starting from slightly different initial data or using slightly different parameters. We can thereby get a probabilistic prediction of the ocean currents, which again can be used for ensemble-based forecasts of drift trajectories.

Figure 1.2 shows the difference between a single deterministic forecast to the left and an ensemble-based probabilistic forecast to the right for a drift trajectory over 72 hours. The ensemble forecast is here run with 50 ensemble members. Each trajectory is plotted with a light blue line, whereas the dark blue lines represent the ensemble mean trajectories for the three drifters. The probabilistic forecast reveals that there is a considerable uncertainty in how far the two lowermost drifters will drift, whereas the uppermost drifter has a much lower variance.

As in operational ocean forecasts, we aim to assimilate relevant observations of the ocean into the ensemble through suitable data-assimilation techniques. Since the ocean evolves through nonlinear dynamics, and since there are few available observations, nonlinear data-assimilation techniques such as particle filters appear attractive [94]. Several new particle filters have been proposed lately [95], and herein we will explore one of those new methods with respect to the problem of drift trajectory predictions.

In summary, our approach is to trade physical complexity with bet-

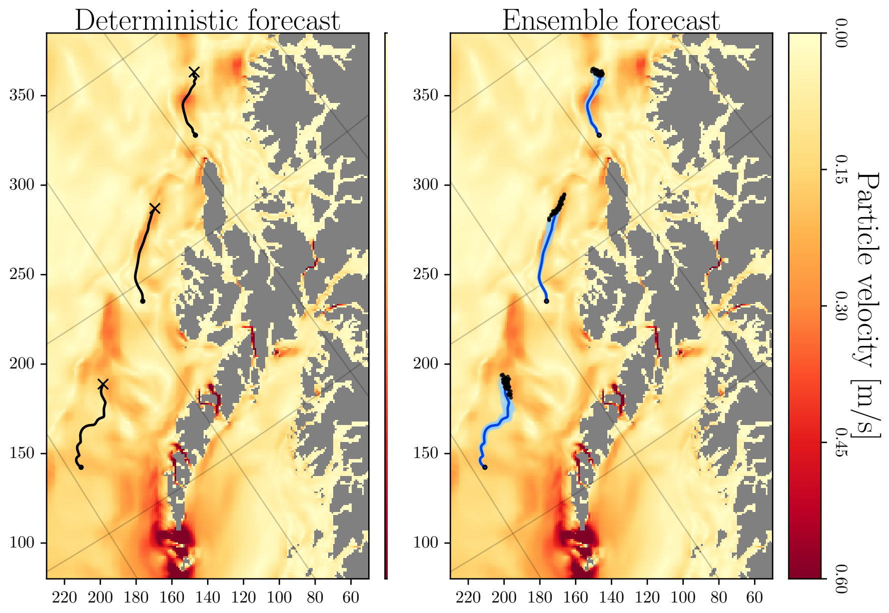


Figure 1.2: Examples of deterministic (left) and probabilistic ensemble-based (right) forecasts for drift trajectories for three drifters located in Northern Norway. Each ensemble member is plotted with a light blue trajectory, whereas the darker blue trajectories represent ensemble means. The background shows the water velocity for the initial conditions, the axes are in km relative to the computational domain, and north is towards the upper left.

ter uncertainty quantification through an ensemble of simplified models and nonlinear data-assimilation techniques. This approach is not meant to replace the drift trajectory forecasts used operationally today, but rather serve as a complementary solution. Furthermore, our algorithms and simulation frameworks can be run on modern commodity-level GPUs and powerful laptops, and do therefore not require access to supercomputers. This makes it possible to use our approach to run updated forecasts directly from a vessel using in-situ observations.

The following chapters give brief overviews of each of the three scientific disciplines the thesis is built upon. In Chapter 2, we show how the shallow-water equations can act as a simplified ocean model. We start by giving an overview of the physical model that describes the dynamics of the ocean, before introducing the simplified model and outlining the most common numerical methods for simulating this model. Chapter 3 gives a basic introduction to how the GPU can be used as an an accelerator for high-performance computing in general, and simulation of the shallow-water equations in particular. Finally, Chapter 4 discusses how observations can be incorporated into such a mathematical model using data assimilation. Here, we start from the fundamental Bayes theorem and give an overview of the different data assimilation methods based on it. This leads us to take a closer look at particle filters, which is the data assimilation technique used in this work. Chapter 5 outlines the contributions presented through this thesis. We show how the scientific disciplines in the previous sections are combined for forecasting drift trajectories, and how the different papers written during these three years build on each other for solving the problem. We then discuss the individual papers that make up this thesis. Chapter 6 summarizes the work and points to the opportunities ahead. Part II contains the five scientific papers in full.

Chapter 2

Simplified Ocean Models and Numerical Methods

This chapter aims to give some background on the shallow-water equations used as a simplified ocean model. The equations have a central role in this thesis and are used in all five papers. We start off by giving a brief introduction to the so-called primitive equations, which are the engine for several operational ocean models, before describing how we can simplify them to become the shallow-water equations. Finally, we outline the basic concepts and ideas behind two of the most common classes of numerical methods used for solving them in a rotational frame of reference.

2.1 Modern operational ocean models

Modern operational ocean models [55, 86] are typically based on solving the so-called primitive equations, which are centered around three important balance relations:

- conservation of mass,
- conservation of momentum, and
- conservation of tracer properties.

The first ensures that water can not appear or disappear by itself, and it is referred to as the continuity equation. The Navier-Stokes equations are the basis for describing conservation of momentum of fluids, and in oceanography it is common to apply some relevant approximations: The hydrostatic approximation assumes that vertical momentum depends only on the balance between the gravitational force and the vertical gradient of the pressure; and the Boussinesq approximation, which assumes that variations in density are small compared to density itself and therefore negligible [77]. The Boussinesq approximation is equivalent to assuming that the fluid is incompressible, meaning that the equation for conservation of mass now also conserves volume. The final balance relation conserves salinity and temperature. These properties are transported by the fluid velocity but also influence the fluid motion as they affect the water density, thus creating buoyancy forces which represent the baroclinic signals in the ocean.[1] Finally, to successfully solve the equations, we need to enforce suitable boundary conditions, such as no flow through the ocean floor and a kinematic boundary condition describing the level of the free surface [77].

Some major examples of operational ocean models that are based on the primitive equations are the Regional Ocean Modeling System (ROMS) [86], the Nucleus for European Modelling of the Ocean (NEMO) [55], and the Hybrid Coordinate Ocean Model (HYCOM) [3, 17]. They use sophisticated numerical models for solving the primitive equations, with an emphasis on balancing the conserved variables, resolving the dynamical processes, and obtaining good computational performance. This involves staggered grids, where the discretized variables are defined in different points in space, and time-splitting techniques for resolving both the diffusive and non-diffusive processes. Furthermore, different strategies are used to split the problem into vertical layers, which is a non-trivial challenge considering that the ocean depth changes from several thousand to only a few meters (see discussion of this topic in [86]).

In addition to simply solving the primitive equations, operational models must also be coupled with advanced data assimilation methods. In this way, they can combine yesterday's forecast with the most recent ocean observations to compute an improved starting point for tomorrow's forecast. For example, both ROMS and NEMO are developed with capabilities to run data assimilation using 4D-VAR [60, 59], whereas the TOPAZ model, which is a 12–16 km resolution HYCOM-based coupled ocean and sea ice model, uses ensemble Kalman filters to assimilate observations for the North Atlantic [100]. This, of course, increases the computational load on the system. We will come back to different data assimilation methods in Chapter 4.

As the purpose of operational models is to provide regular ocean fore-

[1] In the ocean, baroclinicity is a state in which the pressure gradient is unaligned with the depth gradient. This can be caused by variations in temperature or salinity, which cause variations in density, which again leads to variations in pressure.

casts, they must run on a regular basis with only a limited amount of time available for providing the results. Computational performance is therefore of uttermost importance to ensure that the forecast is available in a timely matter, i.e., simulations must run faster than the real-world events. Additional benefits from more efficient models are that they allow for finer grid resolutions that can improve the details and accuracy in the forecasts, and that they free up resources that can be used to assimilate more observations into the models and thus obtain improved initial conditions. Another alternative for taking advantage of improved computational efficiency is by generating probabilistic ensemble-based forecasts through running the model multiple times.

Due to the computational complexity of ocean models and finite amount of computational capacity, it is infeasible to have both high grid resolution and global coverage, and most operational systems therefore concentrate on specific regions. Still, it is important to model changes in the ocean state that enter the domain during the simulation time range, such as tides or storm systems. A common technique for this is to have local or regional models with fine grid resolution nested within other models covering a larger domain with coarser resolution. As an example, two of the operational models for Norway, NorKyst-800 and NorShelf, both use nested boundary conditions from the forecasts provided by TOPAZ [79].

In two of the papers in this thesis, we use the NorKyst-800 model system [2] as a reference solution (Paper IV) or to simulate a true ocean state that we wish to predict (Paper V). It runs the ROMS model on a domain covering the complete Norwegian coast. It uses a horizontal grid with resolution of 800 m $\times$ 800 m and 42 vertical layers. As an illustration of the computational complexity of such a simulation, computing a single 24 hour forecast of this domain takes 45 minutes when using 256 CPUs[2], and it is the results from such simulations that are used as input for drift trajectory predictions.

2.2 Simplified ocean models

In this work, we aim to investigate the use of simplified ocean models to forecast drift trajectories. The idea is to trade some of the physics to obtain faster simulations and then utilize the enhanced computational performance to run a large ensemble of ocean models to obtain probabilistic

[2]Personal communications with Kai Christensen, Head of Division for Ocean and Ice, Norwegian Meteorological Institute, October 2019.

forecasts.

As our simplified ocean model, we use the shallow-water equations in a rotating frame of reference, which can be derived from the primitive equations by imposing some further approximations and assumptions. First of all, the fluid density is assumed to be uniform everywhere, meaning that the variations in salinity and temperature no longer contribute to the dynamics and can be dropped. This also means that the pressure gradient becomes independent of depth. We further assume that vertical velocities are negligible compared to horizontal, and integrate the equations from the sea floor to the free ocean surface. Since the pressure now only depends on the ocean depth, the equations capture barotropic dynamics only. Examples of barotropic waves are fast gravity-driven waves, such as tides, and slow rotation-driven waves, such as Rossby waves. Baroclinic signals, however, that originate from pressure differences caused by changes in salinity or temperature, are neglected. Still, the shallow-water equations are a classical set of reduced or simplified equations for describing both the ocean and atmosphere. For a detailed derivation of both the primitive and the shallow-water equations, see Røed [77].

The shallow-water equations are given by

$$\begin{bmatrix} \eta \\ hu \\ hv \end{bmatrix}_t + \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{bmatrix}_x + \begin{bmatrix} hv \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix}_y = \begin{bmatrix} 0 \\ fhv \\ -fhu \end{bmatrix} + \begin{bmatrix} 0 \\ ghH_x \\ ghH_y \end{bmatrix}, \tag{2.1}$$

and this set of equations is an example of a nonlinear, first-order, quasilinear, hyperbolic conservation law that conserves the deviation of the sea-surface from an equilibrium state η, and the momentum of the water hu and hv along the x and y axes, respectively. The dynamics is driven by the gravity g over a varying bathymetry, here represented by the equilibrium depth H, and the Coriolis forces due to the Earth's rotation, represented by the Coriolis parameter f. Furthermore, $h = H + \eta$ describes the actual depth of the water column, and the subscripts t, x and y denote derivatives with respect to time and space. Figure 2.1 illustrates the various variables, simplified to one dimension.

By defining the vector of conserved variables $\boldsymbol{q} = [\eta, hu, hv]^T$, we can write (2.1) on vector form as

$$\boldsymbol{q}_t + \boldsymbol{F}(\boldsymbol{q})_x + \boldsymbol{G}(\boldsymbol{q})_y = \boldsymbol{S}_C(\boldsymbol{q}) + \boldsymbol{S}_H(\boldsymbol{q}), \tag{2.2}$$

in which $\boldsymbol{F}$ and $\boldsymbol{G}$ are flux functions in x- and y-direction, respectively, and $\boldsymbol{S}_C$ and $\boldsymbol{S}_H$ are source terms accounting for the Coriolis force and changing

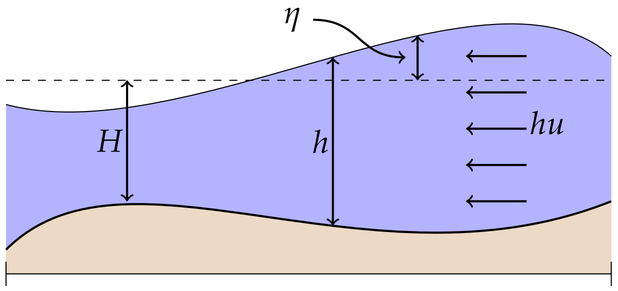


Figure 2.1: The one-dimensional relationship between the equilibrium water depth H, the deviation from mean sea-surface level η, the total water depth h, and the momentum hu.

bathymetry, respectively. Note that for more realistic simulations, it is possible to include more source terms for other external forces. In Paper IV, for example, we also include source terms for the friction along the sea floor and the wind stress that acts on the ocean surface.

It should be noted that (2.1) is not the only way to formulate the shallow-water equations. In the oceanographic community, for instance, it is common to write them as

$$\begin{aligned} \partial_t \boldsymbol{U} + \nabla_H \cdot \left(\frac{\boldsymbol{U}\boldsymbol{U}}{h} \right) + f\boldsymbol{k} \times \boldsymbol{U} &= -gh\nabla_H(h - H), \\ \partial_t h + \nabla_H \cdot \boldsymbol{U} &= 0, \end{aligned} \tag{2.3}$$

in which $\boldsymbol{U}$ is the three-dimensional vector containing the momentum (but with zero in the vertical component), ∇_H is the horizontal gradient operator, and $\boldsymbol{k}$ is the vertical unit vector. Paper I demonstrates that (2.3) is equivalent to (2.1).

2.3 Numerical methods for the shallow-water equations

In total, we have used four different numerical schemes for solving the shallow-water equations in this thesis. Paper I discusses them and compares their abilities to capture different oceanographic phenomena. In the following, we place the schemes within the field of numerical mathematics and quickly review their basic ideas and underlying concepts. We consider the schemes in pairs, as they belong to the following two classes of numerical methods:

Finite-difference methods: Classic approach in which function derivatives are approximated locally by differences in function point values defined over a structured spatial mesh. Still commonly used within oceanography and meteorology.

Finite-volume methods: Solve the problem on a grid by using the conservation law (posed on integral form) to compute fluxes between cells. Commonly used for solving the non-rotational shallow-water equations in the field of hydrology.

The full details for each of the numerical schemes can be found in the references provided when each scheme is mentioned. For a comprehensive introduction to these methods, however, see Røed [77] for finite-difference methods and LeVeque [53] or Toro [93] for finite-volume methods.

Finite-difference methods on C-grids

The finite-difference method is a classical numerical technique for solving partial differential equations and is based on approximating derivatives by the use of Taylor series. For a reasonably smooth function $f(x)$, its value at a point $x_0 + \Delta x$ can be approximated by the Taylor series

$$\begin{aligned} f(x_0+\Delta x) &= f(x_0) + \sum_{n=1}^{\infty} \frac{1}{n!} f^{(n)}(x_0)\Delta x^n \\ &= f(x_0) + f'(x_0)\Delta x + \frac{1}{2} f''(x_0)\Delta x^2 + \mathcal{O}\left(\Delta x^3\right), \end{aligned} \tag{2.4}$$

in which $f^{(n)}(x_0)$ is the n'th order derivative of f with respect to x evaluated at the point x_0, and $\mathcal{O}(\Delta x^3)$ denotes terms of order three or higher in Δx. If we know the function values of f at x_0 and $x_0 \pm \Delta x$, we have three different options for approximating $f'(x_0)$ as illustrated by Figure 2.2. By reordering (2.4) directly, we can get the approximation

$$f'(x_0) = \frac{f(x_0+\Delta x) - f(x_0)}{\Delta x} + \mathcal{O}(\Delta x), \tag{2.5}$$

when the higher order derivatives are included in $\mathcal{O}(\Delta x)$. Similarly, we can use the Taylor series around the point $x_0 - \Delta x$ to express

$$f'(x_0) = \frac{f(x_0) - f(x_0-\Delta x)}{\Delta x} + \mathcal{O}(\Delta x). \tag{2.6}$$

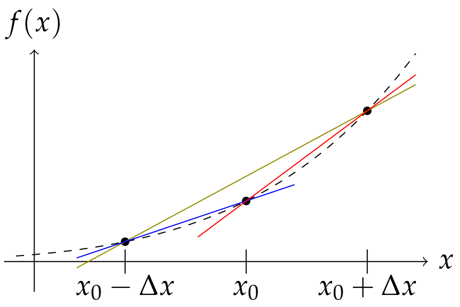


Figure 2.2: Illustration of how the derivative of the dotted function $f(x)$ is approximated at point x_0 by the use of forward differences (red line), backward differences (blue line), and central differences (green line). Note that it is the slope of $f(x)$ that is interesting, and not its function value at x_0.

By neglecting the higher-order terms in (2.5) and (2.6), we call the two expressions the forward and backward difference approximations, respectively. Both are first-order approximations, as the leading reminder term is $\mathcal{O}(\Delta x)$. By subtracting the Taylor series around $x_0 - \Delta x$ from the Taylor series around $x_0 + \Delta x$, we get an approximation for $f'(x_0)$ in which all odd ordered terms cancel, as

$$f'(x_0) = \frac{f(x_0 + \Delta x) - f(x_0 - \Delta x)}{2\Delta x} + \mathcal{O}(\Delta x^2). \tag{2.7}$$

This is called a central difference approximation and it is of second order. These are only the simplest finite-difference approximations available, and higher-order approximations can be made by other combinations of Taylor series.

When using the finite-difference method to solve a partial differential equation, we first discretize the domain onto a grid on which we will solve our equation. Here, we will assume a regular grid with the distance Δx and Δy between grid points in each direction. We then use approximations obtained from Taylor series to estimate the spatial derivatives at each grid point based on values of its neighbouring grid points. Similarly, we discretize t into time steps of length Δt and approximate the time derivatives to be able to evolve the equation forward in time. It is important to take care in the choice of discretization methods, as different approximation schemes have different advantages and disadvantages. We will next briefly mention the two finite-difference methods that we use in Paper I. For an in-depth discussion on the qualities of different finite-difference schemes for oceanographic applications, see Røed [77].

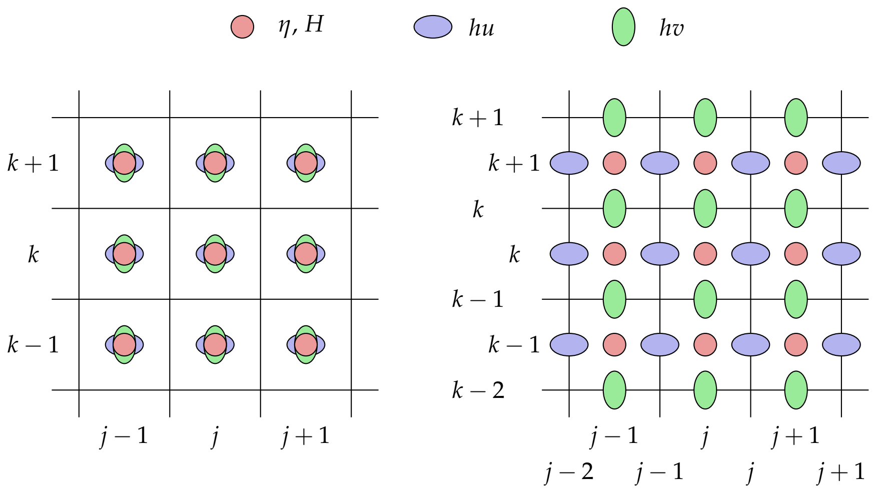


Figure 2.3: Grid with lattice-A structure to the left and lattice-C structure to the right.

In atmospheric and oceanographic applications, it is common to use staggered grids to solve the shallow-water equations in (2.1). Figure 2.3 shows a staggered lattice-C grid along with the non-staggered lattice-A grid [57]. In the lattice-C grid, all of the three conserved variables are located at different grid points, with hu located $\frac{1}{2}\Delta x$ to the right and hv located $\frac{1}{2}\Delta y$ above the corresponding η and H values. The main motivation for using such a staggered grid is that it becomes more convenient to define boundary conditions compared to when using a lattice-A grid. Another advantage is that it becomes very easy to express the equation for mass conservation (first row of (2.1)) using central differences:

$$\eta_{j,k}^{n+1} = \eta_{j,k}^{n} - \frac{\Delta t}{\Delta x}\left((hu)_{j,k}^{n} - (hu)_{j-1,k}^{n}\right) - \frac{\Delta t}{\Delta y}\left((hv)_{j,k}^{n} - (hv)_{j,k-1}^{n}\right). \tag{2.8}$$

Here, we have used a forward difference approximation in time, giving an explicit scheme, which, unlike an implicit scheme, does not involve solving any linear system.

The first finite-difference scheme that we apply in this thesis is the forward-backward linear scheme (FBL) [87], which is used to solve the linearized shallow-water equations:

$$\begin{aligned} \eta_t + (hu)_x + (hv)_y &= 0, \\ (hu)_t + gH\eta_x &= fhv, \\ (hv)_t + gH\eta_y &= -fhu. \end{aligned} \tag{2.9}$$

For each of the equations, we approximate both the derivatives and the other variables according to the grid locations of the time derivative. That is, in the equation for $(hv)_t$ in (2.9), we approximate H as the average of the two H values above and below of each hv location, and represent η_x by central differences by using the same corresponding two point values of η. Similarly, fhu is approximated by the mean of the four hu values surrounding each hv grid point. Using a forward difference approximation in time, the last line of (2.9) thereby becomes

$$(hv)^{n+1}_{j,k} = (hv)^n_{j,k} - \Delta t \overline{fhv}^n_{j,k} + \frac{\Delta t}{2\Delta y} g \left(H_{j,k+1} + H_{j,k}\right)\left(\eta_{j,k+1} - \eta_{j,k}\right), \quad (2.10)$$

in which

$$\overline{fhv}^n_{j,k} = \frac{1}{4}\left(f_k (hv)^n_{j,k} + f_{k+1}(hv)^n_{j,k+1} + f_k (hv)^n_{j-1,k} + f_{k+1}(hv)^n_{j-1,k+1}\right). \quad (2.11)$$

Here, we have discretized the Coriolis parameter based on an assumption that the y-axis points towards north, but as exemplified in Paper IV, this is not always the case. The discretization for $(hu)_t$ follows the same approach as for hv_t, whereas the equation for η_t uses the scheme in (2.8). Note that all spatial approximations are based on second-order central differences. To solve (2.9) in time, the FBL scheme solves the three equations in turn, while always using the most recent values of each variable. That is, we start by obtaining $(hu)^{n+1}$ from $[(hu)^n, (hv)^n, \eta^n]$, then $(hv)^{n+1}$ from $[(hu)^{n+1}, (hv)^n, \eta^n]$, and finally η^{n+1} from $[(hu)^{n+1}, (hv)^{n+1}, \eta^n]$. The first step is therefore a forward difference, whereas the last step becomes a backward difference. This means that the full scheme is of first order in time.

The next scheme is a classic leap-frog scheme called centered-in-time, centered-in-space (CTCS) (see [76]). It solves the full nonlinear equations in (2.1) using central differences in space. This is similar to the FBL scheme, but the CTCS scheme becomes more complicated and involves a larger number of grid points due to the nonlinear terms. In time, all three equations are solved independent of each other with central differences, meaning that CTCS is of second order in both time and space. As an example, the equation for η becomes

$$\eta^{n+1}_{j,k} = \eta^{n-1}_{j,k} - \frac{2\Delta t}{\Delta x}\left((hu)^n_{j,k} - (hu)^n_{j-1,k}\right) - \frac{2\Delta t}{\Delta y}\left((hv)^n_{j,k} - (hv)^n_{j,k-1}\right). \quad (2.12)$$

A drawback for the CTCS scheme is that it is known to suffer from nonlinear instabilities [69, 74, 77]. It is therefore common to add an *eddy viscosity* *term* that introduces artificial diffusion to dampen accumulation of short waves. This term depends on the grid size, and is therefore in practice treated as a tuning parameter.

These two methods represent the traditional numerical models used in meteorology and oceanography, and the FBL scheme was, for instance, used for storm-surge predictions in the end of the 1970s [56]. Even though today's operational methods rely on more complex mathematical models together with more complex numerical discretizations, these methods are relatively inexpensive to run and therefore still of interest in the context of ensemble predictions using simplified models. For a full description of the two schemes, see Røed's book [77] or report [76].

In general, the main drawback for the FBL and CTCS schemes is that they break down in the presence of discontinuities. The methods have nonetheless been popular in oceanography and meteorology, as discontinuities rarely appear in these applications. In other applications that can be modelled by shallow-water equations, such as simulations of tsunamis or dam breaks, it is essential to capture the development of discontinuities accurately. These applications have been the motivation for a whole class of so-called *high-resolution* finite-volume schemes. We therefore turn our attention to one recently suggested scheme that has been developed for oceanographic applications.

High-resolution finite-volume methods on A-grids

Finite-volume methods are closely related to the general expression of a hyperbolic conservation law, such as (2.2). When deriving conservation laws, it is common to consider a control volume and make sure that the change of mass (or any other conserved physical quantity) within the control volume matches the mass that travels through the volume boundary (in the form of flux terms) and mass that appears/disappears internally in the volume (through source terms) within a short time span.

In the formulation of finite-volume methods, we discretize the spatial domain into grid cells of finite size and let each cell represent its own control volume. For the methods considered herein, we use a regular Cartesian grid with cells sizes $\Delta x \times \Delta y$, and discretize the conserved variables $\boldsymbol{q}$ from (2.2) in cell (j,k) by cell average values $\boldsymbol{Q}_{j,k}$. Note that this implies a non-staggered lattice-A grid. Next, we aim to discretize the fluxes on the cell faces, and use the fact that the flux terms in $\boldsymbol{F}$ only contribute to changes across the cell faces with normal vectors in x-direction, and similarly, the fluxes in $\boldsymbol{G}$ only contribute to changes across faces with normal vectors

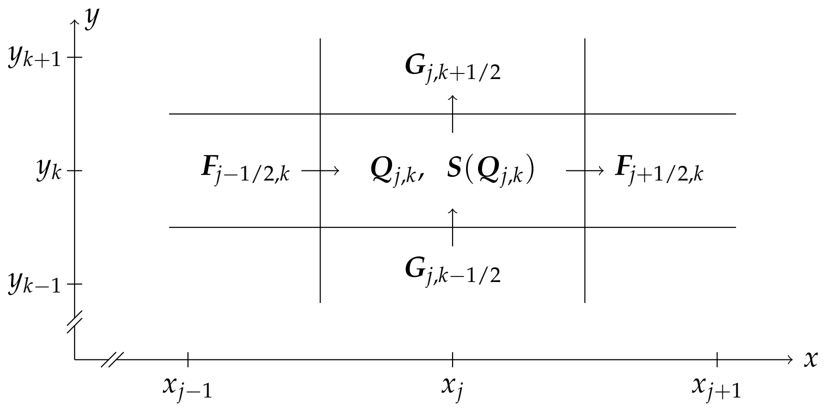


Figure 2.4: Cell values $\boldsymbol{Q}$, source terms $\boldsymbol{S}$ and fluxes $\boldsymbol{F}$ and $\boldsymbol{G}$ for finite-volume methods defined on a regular Cartesian grid.

along the y-axis. We can therefore write (2.2) in semi-discretized form as

$$\frac{\partial \boldsymbol{Q}_{i,j}}{\partial t} = -\frac{\boldsymbol{F}_{i+1/2,j} - \boldsymbol{F}_{i-1/2,j}}{\Delta x} - \frac{\boldsymbol{G}_{i,j+1/2} - \boldsymbol{G}_{i,j-1/2}}{\Delta y} + \boldsymbol{S}(\boldsymbol{Q}_{i,j}), \tag{2.13}$$

with both source terms combined into $\boldsymbol{S}$, as illustrated in Figure 2.4.

Herein, our primary interest is in so-called high-resolution schemes, which offer second or higher order spatial accuracy on smooth parts of the solution, and at the same time are free of the spurious oscillations that are typically observed near discontinuities in classical finite-difference or finite-volume schemes. Many high-resolution methods are constructed according to the so-called *REA* algorithm. Starting from the cell-averaged discrete values, we first *reconstruct* a piecewise polynomial approximation inside each cell. We then set this approximation as initial data and *evolve* the conservation equation forward in time, exactly or approximately, before we *average* the resulting function over the grid so that our representation of the unknown solution again is a set of cell averages. There are, however, large differences between the various schemes in how they reconstruct point values, compute fluxes, and evolve the approximate solution [53, 93].

In this work, we use two different high-resolution schemes of second order. Both are based on the same philosophy and follow the steps illustrated in one dimension in Figure 2.5. As all REA-type schemes, they start from the discretized values $\boldsymbol{Q}_j$ in (b), which represent cell averages of the continuous state shown in (a). The bathymetry is, however, represented by

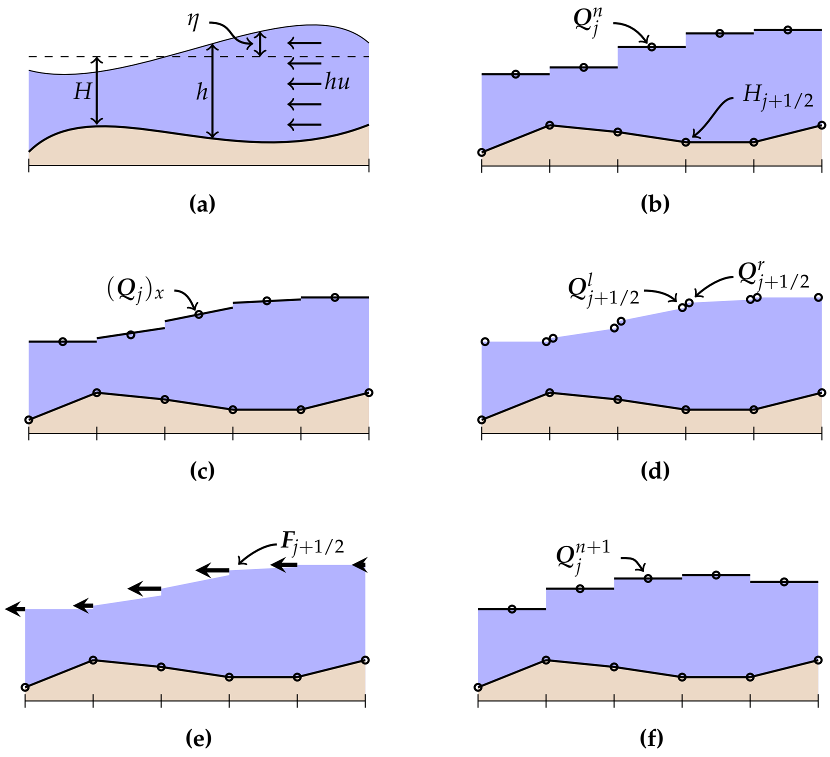


Figure 2.5: Illustrative description of the finite-volume schemes used in this work in one dimension. (a) Continuous variables with (b) their piece-wise constant, cell-averaged equivalents. (c) Reconstruction of the slope of Q, leading to (d) a reconstruction of point values at the cell faces and (e) fluxes on the cell faces. (f) New average cell values are then found at the next time step. This illustration is based on figures from Brodtkorb et al. [12].

a piecewise bilinear function defined by values at the intersections $H_{j+1/2}$ (or $H_{j+1/2,k+1/2}$ in two dimensions).

In Figure 2.5 (c), we compute the slopes $(Q_j)_x$ of a piecewise linear reconstruction, with the aim of finding the state values at the face as seen from each cell. This must be done without introducing spurious oscillations in the solution. Straightforward use of backward, forward, or central differences to compute the slopes (see Figure 2.2), may induce overshoots or undershoots in the reconstructed functions near discontinuities or extreme points, which may develop into spurious oscillations. Instead of using the finite-difference candidates directly, we therefore use a nonlinear

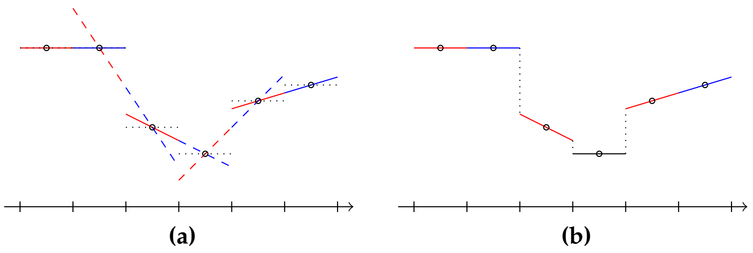


Figure 2.6: Applying a non-oscillatory limiter function to the forward (red) and backward (blue) finite-difference approximations for the slopes in the given points. All candidates are shown in (a), whereas (b) shows the resulting slopes only. Note that both candidates lead to oscillations in the fourth cell, and we therefore chose a zero-slope instead.

combination to ensure that the total variation of the reconstructed function does not increase, i.e., ensure the so-called total-variation-diminishing (TVD) property [36, 92]. The nonlinear function is commonly referred to as a limiter. Figure 2.6 shows an example of a limiter using the forward and backward approximations in red and blue, respectively. Here, we see that the dashed approximations lead to over- or undershoots compared with the cell averages, whereas the solid lines do not. For the lowermost cell (number four from the left), both forward and backward approximations give undershoots, and we use a zero slope instead.

Using the slopes $(\boldsymbol{Q}_j)_x$, we find the state values at each side of each face as seen from the left and right, denoted in Figure 2.5 (d) by $\boldsymbol{Q}^l_{j+1/2}$ and $\boldsymbol{Q}^r_{j+1/2}$, respectively. The fluxes $\boldsymbol{F}_{j+1/2}$ in (e) are computed through a central-upwind scheme [50],

$$\begin{aligned}\boldsymbol{F}_{j+1/2} = &\frac{a^+_{j+1/2}\boldsymbol{F}\left(\boldsymbol{Q}^l_{j+1/2}\right) - a^-_{j+1/2}\boldsymbol{F}\left(\boldsymbol{Q}^r_{j+1/2}\right)}{a^+_{j+1/2} - a^-_{j+1/2}} \\ &+ \frac{a^+_{j+1/2}a^-_{j+1/2}}{a^+_{j+1/2} - a^-_{j+1/2}}\left(\boldsymbol{Q}^r_{j+1/2} - \boldsymbol{Q}^l_{j+1/2}\right),\end{aligned} \tag{2.14}$$

in which a^+ and a^- are the largest positive and negative wave speeds at the interface, given by

$$\begin{aligned}a^+_{j+1/2} &= \max\left\{u^l_{j+1/2} + \sqrt{gh^l_{j+1/2}}, u^r_{j+1/2} + \sqrt{gh^r_{j+1/2}}, 0\right\}, \\ a^-_{j+1/2} &= \min\left\{u^l_{j+1/2} - \sqrt{gh^l_{j+1/2}}, u^r_{j+1/2} - \sqrt{gh^r_{j+1/2}}, 0\right\}.\end{aligned} \tag{2.15}$$

The fluxes in (2.14) are therefore found by using the local one-sided directions of the flow.

The schemes extend directly to two dimensions as the fluxes in $\boldsymbol{G}$ can be found independent of the fluxes in $\boldsymbol{F}$. Finally, to get updated cell-average values, we use a strong stability-preserving Runge-Kutta method for the time evolution [31]. The simplest such method, which is used to illustrate the new cell averages in (f), is the standard first-order forward Euler

$$\boldsymbol{Q}_{j,k}^{n+1} = \boldsymbol{Q}_{j,k}^{n} + \Delta t \boldsymbol{R}(\boldsymbol{Q}^n)_{j,k}, \tag{2.16}$$

in which $\boldsymbol{R}(\boldsymbol{Q})$ denotes the right-hand side of (2.2). Throughout this work, however, we use the second-order method

$$\begin{aligned} \boldsymbol{Q}_{j,k}^{*} &= \boldsymbol{Q}_{j,k}^{n} + \Delta t \boldsymbol{R}(\boldsymbol{Q}^n)_{j,k}, \\ \boldsymbol{Q}_{j,k}^{n+1} &= \frac{1}{2}\Big(\boldsymbol{Q}_{j,k}^{n} + \Big(\boldsymbol{Q}_{j,k}^{*} + \Delta t \boldsymbol{R}(\boldsymbol{Q}^*)_{j,k}\Big)\Big). \end{aligned} \tag{2.17}$$

Note that the second-order scheme in (2.17) can be described as the average between $\boldsymbol{Q}^n$ and $\boldsymbol{Q}^{n+2}$, if $\boldsymbol{Q}^{n+2}$ is found using the first-order scheme in (2.16).

Both the finite-volume schemes we use herein are *well-balanced* with respect to certain steady-state solutions. A steady state is described by $\boldsymbol{q}_t = 0$, meaning that the state (recall that $\boldsymbol{q} = [\eta, hu, hv]^T$) does not change over time. A lake at rest, with a perfectly flat and still surface with no water movements, is a trivial example of a steady-state. As long as there are no external forces, this situation should not change, regardless of how complex the bathymetry is. Well-balanced schemes are schemes that maintain such steady-states, and the well-balanced property is therefore important to avoid spourious oscillations. Numerically, we obtain this property by ensuring that the flux reconstructions and the discretizations of the source terms are perfectly balanced for the steady-state solution.

The scheme proposed by Kurganov and Petrova [49] (building on [48] and used in Paper IV under the abbreviation KP) for non-rotational shallow-water problems is designed in this way and is well-balanced for the lake at rest steady-state solution in the presence of discontinuous bathymetry. The other finite-volume scheme used in this thesis, was proposed by Chertock et al. [19] (CDKLM). It is well-balanced with respect to the geostrophic balance, which is a non-trivial rotating steady-state solution described by

$$\begin{aligned} fhv - gh\frac{\partial \eta}{\partial x} &= 0, \\ -fhu - gh\frac{\partial \eta}{\partial y} &= 0. \end{aligned} \tag{2.18}$$

An example of this steady-state solution is a bump with a rotational momentum that balances the gravitational forces and thereby traps the water, avoiding a radial wave. To balance this steady state, CDKLM reconstructs η based on the change in potential energies, in which (2.18) plays an important role. Due to this special reconstruction, this scheme is very well suited for oceanographic applications, and in Paper IV we demonstrate how the scheme can be extended to run simulations on real-world operational domains and initiated by operational ocean forecast data.

In summary, high-resolution finite-volume methods have several qualities that are attractive for oceanographic applications. Our motivation for studying such schemes in this thesis, is therefore to assess whether a scheme such as CDKLM can compete with the traditionally used finite-difference schemes for oceanographic physics. One advantage of the high-resolution schemes is that they are more robust, meaning that if a discontinuity should develop, the schemes do not break down. Another is that they do not require any tuning parameters, such as the eddy viscosity term required by CTCS. Furthermore, finite-volume methods can be used efficiently for adaptive mesh refinement [83], so that we can run simulations with increased resolution in areas where interesting features evolve. This would also enable us to run simulations with high spatial resolution in coastal areas while having lower resolution out in the open sea.

Chapter 3

High-Performance Computing using GPUs

One reason for choosing the shallow-water equations as the basis for a simplified ocean model, is that they can be simulated very efficiently on massively parallel hardware, such as GPUs (see, e.g., [33, 12, 24, 9, 67, 71, 39]). In this chapter, we give a brief introduction to the GPU by following its development from a pure specially designed graphics processor, via high-performance accelerator for massively parallel computations, and now emerging as a specially designed machine learning number crusher. We continue by demonstrating how the finite-volume methods from Section 2.3 can take advantage of the fine-granular parallelism, as this is what enables us to run large ensembles of simplified ocean models. Throughout the work with this thesis, we have taken great advantage of using a programming environment that enables us to develop highly efficient code for performance-critical algorithms, but still supporting rapid prototyping, script-based pre- and post-processing, and flexible programming of the simulation program flow. We therefore finish this chapter with a discussion of modern code development for high-performance computing (HPC) applications.

3.1 General-purpose computations on GPUs

Whereas the central processing unit (CPU) is the general-purpose brain of the computer and is designed to execute any kind of instructions, the GPU was initially designed specifically for graphics processing. The most important driving factor for the development of the GPU has therefore been the gaming industry, meaning that it was important to create a processor that as efficiently as possible could update the color values of all pixels in a scene according to the player's actions. Important design criteria was therefore

- that high throughput was more crucial than high numerical precision, since values eventually would be represented by colors only;
- a high degree of parallelism, as the updated color of every pixel could be computed independently from the updated colors of all other pixels;
- parallel processing of the type single-instruction, multiple-data, since the algorithm for updating pixel values often was the same across a large number of pixels;
- that there was a limited set of operations and data types that are important for a large number of graphics processing algorithms.

This resulted in a device consisting of a large number of computational cores with efficient single-precision arithmetics and a limited instruction set, high memory bandwidth, and cores that operate in groups for efficient data management and execution of instructions.

By targeting the gaming market, the GPUs became inexpensive compared to other computational hardware with similar throughput capabilities. This triggered the question whether it would be possible to use GPUs for other kinds of computationally intensive problems as well. Following Larsen and McAllister [51], who first demonstrated that the GPU could be used for matrix-matrix multiplication in 2001, a wide effort was made to take advantage of the GPU for various scientific computing problems. Hagen et al. [33] presented in 2005 the first GPU-based implementation of an explicit high-resolution finite-volume scheme for solving the shallow-water equations. Simultaneously, a large number of programming languages were proposed to facilitate general-purpose computations on the GPU (for details, see [64, 65, 10] and references therein). The two programming platforms CUDA [62, 84] and OpenCL [44] were introduced in 2007 and 2009, respectively, and became the best alternatives. They offered mature, portable, and stable programming environments for general-purpose GPU (GPGPU) computing, and thereby accelerated the research on GPU-accelerated numerical algorithms.

Along with the popularity and success of GPU computing, a new class of GPUs, such as Nvidia's Tesla series, targeting the HPC market has been released. The main advantage over commodity-class gaming GPUs is that the HPC GPUs offer a higher capacity for processing double-precision arithmetics, which is considered a requirement for many problems in scientific computing. Their main disadvantage is a considerably higher price, driven by the fact that the HPC market operates on a different monetary scale than the gaming market. The HPC GPUs have become a huge success, as can be seen by their representation on the Top 500 list, which ranks the best supercomputers in the world with respect to number of floating-point operations per seconds (FLOPS) [58].

Many algorithms within machine learning and artificial intelligence depend on processing large amounts of training data to build models that can be used to make inference from new unknown data. The GPU has played a central role in the recent development boost in these fields by enabling the high computational throughput that has made this development feasible (see, e.g., [47]). Since such algorithms can be further accelerated by the use of low-precision arithmetics [32, 22], the latest Nvidia V100 model has now been designed with new tensor cores especially designed for accelerating machine-learning algorithms [63]. The GPU has thus transformed from a special-purpose graphical processor, via a general-purpose accelerator, and now towards a special-purpose machine-learning number cruncher.

GPUs also have a role to play within the field termed as green computing. Even though GPUs typically have a higher power consumption than CPUs,[1] GPUs are able to deliver more FLOPs per watt and have therefore been shown to increase the energy efficiency compared to multi-core CPUs (see, e.g., [42, 70, 25]). This is also reflected by the Green 500 list [29], which is released in parallel with the Top 500 list, ranking the supercomputers according to their power efficiency. This list is typically dominated by supercomputers that use GPUs, and on average, we find GPUs in eight of the top ten entries in all six lists published from 2017 to 2019, whereas the same number in the regular Top 500 list is only four. In Paper II, we investigate this aspect of GPU computing further by comparing the power efficiency between CUDA and OpenCL, and between a range of different GPUs.

3.2 The shallow water equations on GPUs

Simulation of hyperbolic conservation laws using explicit numerical schemes is a perfect application for general-purpose GPU computing [34], and sev-

[1]Using the CPUs and GPUs found in the new supercomputer system Saga at NTNU as an example, we can find that the Intel Xeon-Gold 6126 CPUs have a power draw of 125 W, whereas the the Nvidia P100 GPUs draw 250 W.

eral papers have been written on efficient simulation of the shallow-water equations on GPUs (see, e.g., [33, 12, 24, 9, 67, 71, 39]). We will shortly explain the general implementation strategy, but first we have to take a quick look at the programming model for CUDA and OpenCL.

CUDA and OpenCL are in many ways very similar to each other and are both based on the same programming model. The terminology, however, differs, and in the following we will use that of CUDA. Functions implemented to run on the GPU are called *kernels* and are executed on a 1D, 2D or 3D *grid* of threads, that again are grouped in *blocks*. You can think of the blocks as a domain-decomposition of the grid. Threads within the same block are executed in parallel and can communicate and share data through the *shared memory*, which can be thought of as a programmable cache. For instance, if multiple threads need to read an overlapping data block from the global GPU memory, it is in general more efficient that the threads cooperate to read the data into shared memory first and afterwards access all required data individually from the shared memory, instead of all threads reading all data from the global memory. A grid can consist of a lot more threads than the number of available GPU cores, and the blocks are therefore executed asynchronously on the GPU with as many blocks in parallel as the GPU's resources permit. Limiting factors can be the number of physical computing cores, number of registers, amount of shared memory, etc. A consequence of this is that it is not possible to synchronize between threads in different blocks during a kernel execution. Synchronization across all threads can therefore only be made in-between different kernel launches.

To efficiently implement the shallow-water equations on GPUs, we use a direct translation from a finite volume grid cell to a GPU thread, or similarly from a finite-difference grid point to a thread. This means that in our shallow-water implementations, each thread is responsible for updating a single value for each of η, hu and hv. In Figure 3.1, we show the data dependencies for each of the steps in a single iteration using any of the two finite-volume schemes from Section 2.3 in one dimension. We use a first-order Runge-Kutta method for simplicity. Contrary to in Section 2.3, we start from the result and follow the steps, and thus the data dependency, in reverse order. We compute the final cell averages $\boldsymbol{Q}^{n+1}$ by using two values of $\boldsymbol{F}$ defined on the faces. These are again found independently from the reconstructed point values $\boldsymbol{Q}^l$ and $\boldsymbol{Q}^r$, which again depend on the cell values $\boldsymbol{Q}^n$ and slopes $\boldsymbol{Q}_x$ in the two cells on each side of the face. The slope $\boldsymbol{Q}_x$ finally depends on the value $\boldsymbol{Q}$ in its own cell and in the two adjacent cells. The total data dependency, often called *the stencil*, for $\boldsymbol{Q}_j^{n+1}$ is therefore the

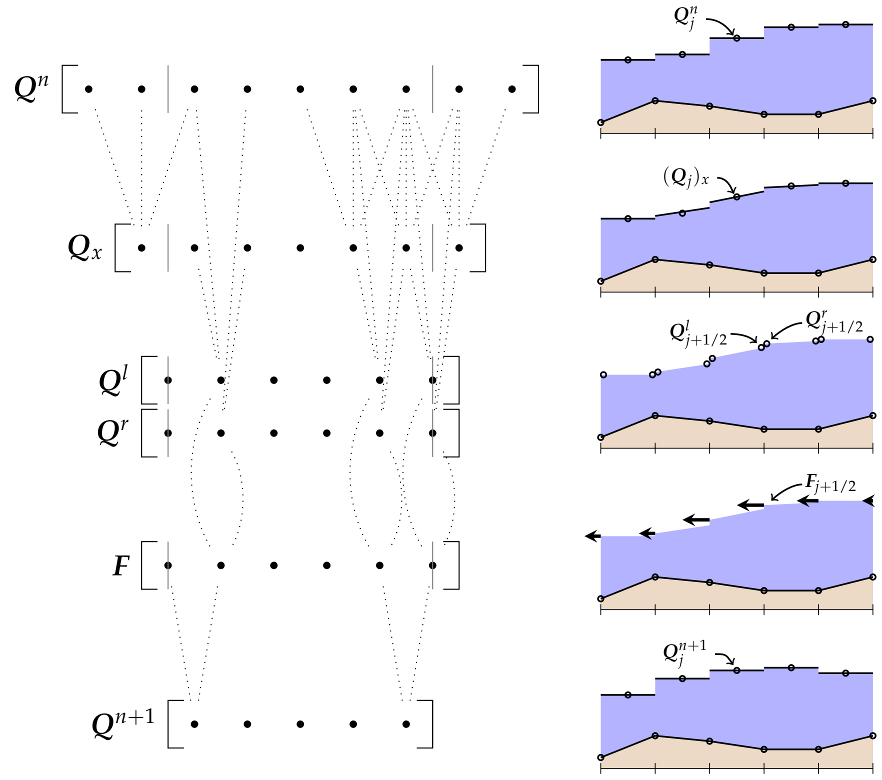


Figure 3.1: Shared memory usage for a GPU implementation of the one-dimensional shallow-water equations using high-resolution finite-volume schemes. To the right in the memory buffers we see the complete data dependency for a single output value and to the left we see the dependency for each computational step.

five values from Q^n_{j-2} to Q^n_{j+2}.

In addition to the data dependencies, Figure 3.1 exposes the parallel potential in each of the computational steps. For each step, each computed value can be found independently from all of the other computed values, even though their data dependencies overlap. This is exactly the kinds of operations for which GPUs excel: Parallel operations on aligned data with common mathematical operations and data dependency patterns.

We split the domain we want to simulate into blocks consisting of b_x threads, so that each will be responsible for computing b_x values of Q^n. In the example in Figure 3.1, we have $b_x = 5$. All b_x threads in the block cooperate to read the $b_x + 4$ values of Q^n into the shared memory, so that all these values are available to all threads. When all values are read, the threads can cooperate to compute $b_x + 2$ values for $\boldsymbol{Q}_x$, which are then written to shared memory. The threads can then use their neighbour's results to compute $\boldsymbol{Q}^l$ and $\boldsymbol{Q}^r$, followed by the $b_x + 1$ values of $\boldsymbol{F}$. Finally, each thread computes its dedicated $\boldsymbol{Q}^{n+1}$ value and write it to the main global memory on the GPU. Note here, that we need to synchronize the threads between each computational step (except between $\boldsymbol{Q}^l$ and $\boldsymbol{Q}^r$), so that a thread does not start evaluating the next step before its data dependencies are ready. There are, however, no need for synchronization between blocks, meaning all these operations can be programmed within a single kernel function.

Figure 3.2 shows the CUDA block structure in the two-dimensional case. Here, the data dependency for a single cell takes the shape of a plus-sign, reaching two cells in all four directions as illustrated by the two examples in red. We have highlighted two of the blocks in blue with the combined data dependency for each of them in a lighter color, showing the overlapping buffers read by each block. Since it is expensive when threads in the same block follow different instruction branches due to conditional branching, we want to impose the same mathematical operations on cells close to the boundary as we do elsewhere. We achieve this by using *ghost cells*, shown as the two outermost cell layers in Figure 3.2, and we initialize them so that the fluxes on the actual boundary evaluate to their appropriate boundary conditions. The strategy outlined here is also highly suitable for the finite-difference schemes discussed in Section 2.3, and their stencils are shown in Figures 3 and 4 in Paper I.

Numerical precision

The use of double-precision floating-point numbers is the default choice when developing numerical simulation software. However, if we can use single-precision arithmetics instead, both the storage and computational requirements are reduced by a factor two. This applies for applications using the CPU and high-performance GPUs.[2] When using commodity-level GPUs, however, the performance gain is considerably higher. For instance, the Nvidia GeForce GTX 780, which has been used extensively during the work for this thesis, has 24 times better theoretical peak per-

[2]Double-precision floating-point numbers use eight bytes to represent numbers, contrary to only four bytes which are used for single-precision. Double-precision therefore increases memory requirement, but also improves the accuracy of how well numbers are represented by the computer.

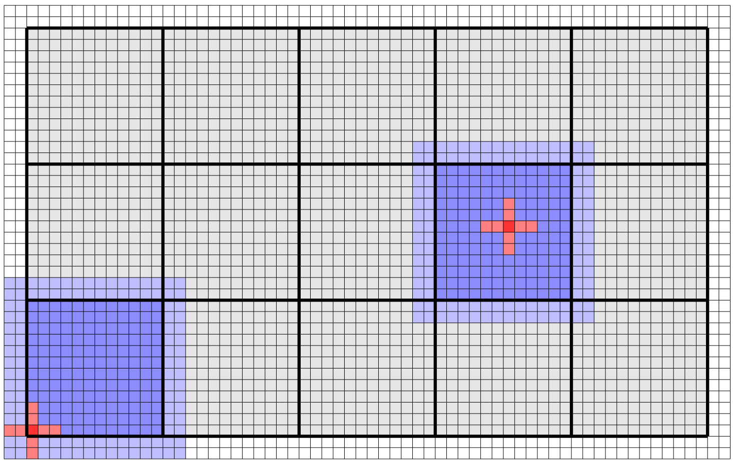


Figure 3.2: The two-dimensional block structure for shallow-water simulation on the GPU. The dark blue areas show the computational domains for two different blocks, with their data dependencies in light blue. The stencil for solving the equation in a single cell is shown in red.

formance with single-precision compared to double-precision. By implementing the numerical algorithms so that precision-related rounding errors are smaller than the discretizations errors in the our numerical approximations, we manage to increase the computational performance significantly. Brodtkorb et al. [11] have previously shown that single-precision is sufficient for simulations of the shallow-water equations on non-rotating domains for dam-break simulations, and we have herein built on that experience.

Recently, the use of single-precision arithmetics has gained more attention also within the atmospheric and oceanographic communities. For instance, Vana et al. [96] showed that the ECMWF's Integrated Forecast System could gain 40% computational efficiency from using single-precision without any reduction in the forecast accuracy. Furthermore, Hatfield et al. [37] demonstrated that by first reducing the memory requirement by using reduced-precision arithmetics for an ensemble forecast, and then reinvesting the now available memory into a larger ensemble, they were able to improve their forecast skills of a simplified atmospheric model.

3.3 Efficient development of modern HPC code

Conventional wisdom says that if you want to develop computationally efficient numerical software, you need to write your code in a strongly typed, compiled programming language, such as C/C++ or Fortran. In these languages, you are required to specify data types of variables and memory allocations, and you thereby have a lot of control over low-level features that are crucial for optimal performance. Furthermore, as compilation is a stand-alone operation separate from running the program, the compiler can carry out additional optimizations while translating the code into machine-readable instructions. The downside of using such programming languages is that you need to specify all these details, making code development both cumbersome and error prone. Also, testing of new functionality can be tedious, since you need to recompile and run a complete program to see if the new functionality works as expected.

On the other hand, higher-level, weakly typed and interpreted languages, such as Python, offer a more flexible programming environment. For instance, you do not need to specify data types for your variables, and you can test separate lines of code on the fly, since interpreted languages can be executed line by line. This freedom comes in general at the cost of performance. Since the code is interpreted on the fly, there is limited time for the compiler to carry out any extra optimizations. Furthermore, since we do not have direct control of data types and memory allocations, the memory management will be handled by the Python interpreter, meaning that for programs where this is important for performance, we have fewer tuning possibilities.

An important thing to keep in mind when it comes to efficient numerical software, is that not all parts of the code need to have the same level of optimization. Most often, the majority of the execution time is spent on a limited part of the code only. Typically, this is the so-called *inner loop*, in which we repetitively solve the model equations through the numerical scheme. All simulations we consider in this work are of this nature, and the performance of these simulations depends almost completely on the performance of the inner loop. We can therefore use a compiled language with access to low-level features for the performance-critical inner loop and Python for orchestrating the program flow, and thus get the best of both worlds. We use CUDA to write efficient simulation code for the of the shallow-water equations, and access the CUDA kernels along with the complete CUDA API from Python through the PyCUDA package [45], which gives us full control of all memory allocations and computations carried out on the GPU. We then write the simulation loop and all the pre- and post-processing of simulation data in Python with all the advantages that an interpreted environment has to offer.

In addition to using Python, we have developed all code in this project by using Jupyter Notebooks [46] as a rapid prototyping platform. The notebooks offer a programming environment that resembles a lab journal, and it supports the writing of code and text in the same interactive document along with figures generated through Python's Matplotlib package [43]. An efficient software-development technique, in my experience, is to prototype new functionality directly in the interactive notebook environment until the code produces the desired results. We can then clean the code while ensuring it is still correct within the notebook, and finally move the code into a suitable Python module. Similarly for GPU code, it can be efficient to prototype a serial version in Python to obtain a correct but slow version, before turning to CUDA to implement the proper parallel version of the code. The Python version will thereafter not be used for simulations, but is still a valuable asset for regression testing and debugging purposes. In Paper II, we include a more detailed discussion concerning this approach for developing efficient GPU-accelerated simulators.

Chapter 4

Data Assimilation

Computational simulation of geophysical processes will always contain errors. From a simulation perspective, we know that all mathematical models to some extent represent idealized and simplified descriptions of the real world. We make assumptions that allow us to neglect certain processes and simplify others, just to be able to describe the physics as a set of equations. Next, to simulate the equations, we need to discretize them onto a suitable grid using a numerical method, such as described in Chapter 2.3. This introduces numerical discretization errors, both in the approximations of derivatives and because the solution is only represented by a finite number of values related to the grid. Finally, when running the simulation, the computer can only represent the numbers to a certain precision.

We can keep control of some of these errors. For instance, we know the precision of floating-point numbers used by the simulation software, and in some cases we can get rigorous error bounds on the discretization methods through numerical analysis. We can also check whether or not our simulation complies with the assumptions for the mathematical model we use. What we can not control as easily is the accuracy in the initial conditions when we want to run real-world simulations, and such simulations can never become more accurate than the accuracy of the initial conditions. For oceanographic applications, for instance, it is impossible to obtain an accurate description of the currents, temperature and salinity throughout a typical simulation area at a given point in time. Furthermore, there are uncertainties related to the exact topography of the sea floor, the amount of wind acting on the uppermost layer, and parametrization of processes that are too small to be represented by the computational grid. All we have available, is typically sparse and uncertain observations of the ocean and simulated estimates on the present ocean state.

In this chapter, we look into the techniques known as data assimilation, which combine uncertain observations with uncertain simulations to reduce the overall uncertainty in the results. Some of the description of particle filters in Section 4.3 follows Section 2 of Paper III.

4.1 Uncertainty quantification

In the presence of uncertain initial conditions, boundary conditions, and model parameters, it is hard to know how accurate the results from a single simulation experiment are. Even if we should know the uncertainties of the initial state of the system, it is still not trivial to model how these uncertainties develop in time. The aim of *uncertainty quantification* is to measure the errors in simulation results, and one way to achieve this is through *Monte Carlo* simulations. Instead of running a single simulation, a Monte Carlo simulation consists of running an *ensemble* of many simulations, in which each simulation run uses slightly different random samples of initial conditions and parameters. From the results of all these simulations, we can find the statistically most likely result through the mean of all the simulated states, whereas the uncertainty of the simulation can be represented through the variance of the ensemble members.

In mathematical terms, we use $\boldsymbol{\psi}^n$ to denote the *state vector* at time t_n. The state vector contains all the variables in our simulation, meaning that for the discretized shallow-water model discussed in Chapter 2.3, the state vector will be

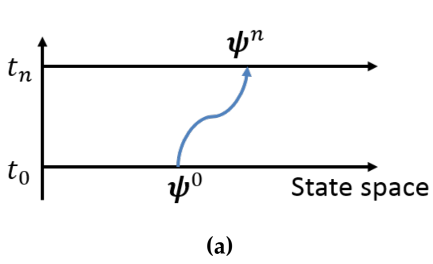
$$\boldsymbol{\psi}^n = [\eta^n_{j,k}, (hu)^n_{j,k}, (hv)^n_{j,k}]^T \in \mathbb{R}^{N_{\psi}}, \tag{4.1}$$

for all indices j, k in the grid. If the computational grid consists of $n_x \times n_y$ cells, the size of the state vector becomes $N_{\boldsymbol{\psi}} = 3n_x n_y$. The ensemble is then a set of state vectors, $\{\boldsymbol{\psi}^n_i\}_{i=1,\dots,N_e}$, in which N_e is the number of ensemble members.

We aim to initialize the ensemble so that it reflects the uncertainty in the initial conditions and model parameters, meaning that the set $\{\boldsymbol{\psi}^0_i\}_{i=1,\dots,N_e}$ becomes a discrete representation of the probability distribution of these uncertainties. We can thereby write the *probability density function* (pdf) of the initial state in terms of the ensemble as

$$p(\boldsymbol{\psi}^0) = \frac{1}{N_e}\sum_{i=1}^{N_e} \delta\left(\boldsymbol{\psi}^0 - \boldsymbol{\psi}^0_i\right), \tag{4.2}$$

in which $\delta(\boldsymbol{\psi})$ is the Dirac-delta function, evaluating to one if $\boldsymbol{\psi} = 0$ and zero otherwise. By using the numerical simulator that represents our



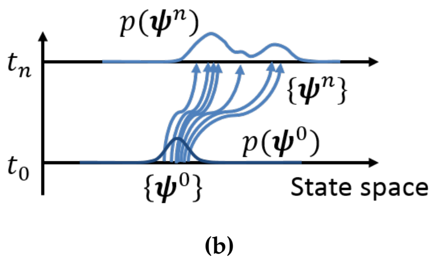


Figure 4.1: The different between (a) a single deterministic forecast and (b) a probabilistic ensemble forecast. Even though the initial conditions have a known and regular pdf, the dynamics in the model can create an irregular pdf for ψ^n.

model, we can evolve each ensemble member independently in time, as

$$\psi_i^n = M\left(\psi_i^{n-1}\right) + \beta_i^n, \tag{4.3}$$

for $i = 1, ..., N_e$. Here, M is the *model operator*, which represents the numerical scheme that evolves the solution from t_{n-1} to t_n, and β_i^n is an optional *model error* which can be used to represent the uncertainty in the model through a random perturbation of the state vector. This lets us express the pdf of the simulation results at time t_n in the same way as (4.2):

$$p(\psi^n) = \frac{1}{N_e}\sum_{i=1}^{N_e} \delta\left(\psi^n - \psi_i^n\right). \tag{4.4}$$

Figure 4.1 shows a deterministic forecast compared to a probabilistic Monte Carlo forecast. Statistical information about the solution can then be obtained from this pdf, such as an estimate for the mean

$$\mathrm{E}\left[\psi^n\right] = \frac{1}{N_e}\sum_{i=1}^{N_e} \psi_i^n, \tag{4.5}$$

and the variance

$$\mathrm{Var}\left(\psi^n\right) = \frac{1}{N_e - 1}\sum_{i=1}^{N_e} \left(\psi_i^n - \mathrm{E}\left[\psi^n\right]\right)\left(\psi_i^n - \mathrm{E}\left[\psi^n\right]\right)^T. \tag{4.6}$$

This enables us to say something about the trustworthiness and uncertainty of the simulation as a whole.

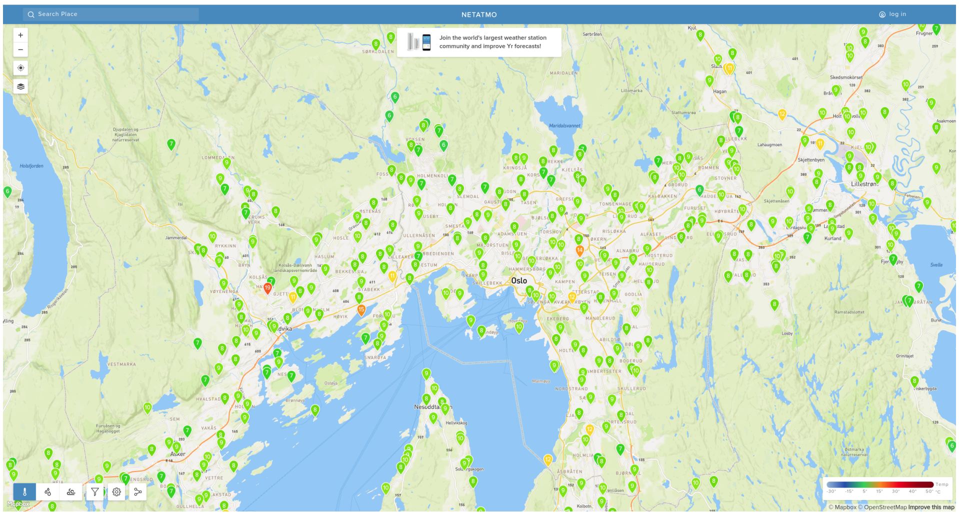

NETATMO
Search Place
log in
Join the world's largest weather station community and improve Yr forecasts!
Oslo
Lillestrøm
Nesoddta
Asker
Temp
© Mapbox © OpenStreetMap Improve this map

Figure 4.2: Temperature observations from Netatmo weather stations in and around Oslo. There are dense observations within the city and in the suburbs, but few in the forests. Even with a large number of observations we only observe parts of the state vector. Screenshot from `https://weathermap.netatmo.com/`.

4.2 The data assimilation problem

For some applications, partial observations of the system state that we simulate become available within the simulation time range. The problem of numerical weather prediction, or simply weather forecasting, is a perfect example of this, as weather stations continuously make measurements (observations) of the atmosphere in the form of ground temperature, precipitation, wind, humidity, etc. The weather stations do, however, only provide observations at certain locations, and then only close to the ground. Even through the use of personal weather stations, e.g., those produced by Netatmo, it is still not possible to observe the full state vector for the atmosphere. Figure 4.2 shows a map that illustrates the density of observations from such weather stations in the Oslo area. There are a large number of weather stations in the populated areas, but no observations of the atmosphere in the forests. This illustrates that even though there are many observations of the atmosphere, these are not sufficient to create complete initial conditions. Furthermore, if we have an observation in a single grid cell, we can still get problems from using that observation directly in our state vector. First of all, we do not necessarily know that the observation is correct, but it could also lead to a situation that is not physically consistent with the surrounding grid values, and this could negatively impact the simulation results in the next time steps.

In contrast to the relatively densely observed atmosphere, there are very few observations of the ocean [35]. An important observational tool in the ocean is the fleet of Argo drifters. They can control their buoyancy but not their horizontal transport, and are therefore deep-diving, passive drifters that collect measurements of the ocean currents and make depth profiles of salinity and temperature [85]. Other observations are made by satellites that measure sea-surface temperature and sea-surface elevation, but only as long as the weather is good [52]; moored buoys, such as the acoustic Doppler current profiler, that observe the current throughout the entire vertical water column at their location [35, 102]; high-frequency (HF) radars, which are installed in pairs at the coast and are used to observe the surface current [73]; and seals with sensors strapped to their heads [80]. In this work, we base our observations on the GPS positions of drifters that passively follow the ocean currents at a constant depth close to the ocean surface, and that can be dropped in the ocean at suitable locations during, e.g., a search-and-rescue mission.

Returning to the mathematics, we denote an observation at time t_n by

$$\boldsymbol{y}^n = H\left(\boldsymbol{\psi}^n_{true}\right) + \boldsymbol{\epsilon}^n. \tag{4.7}$$

Here, $\boldsymbol{\psi}_{true}$ is the true, but unknown, state of the system and H is the observation operator that transforms some of the state variables to the observation vector $\boldsymbol{y}$. We use N_y as the size of $\boldsymbol{y}$. In addition, we have an observation error $\boldsymbol{\epsilon}$. It represents not only the instrumental uncertainty, but also the *representation error*, which says something about how representative the observation is for the underlying state variables. We typically assume $\boldsymbol{\epsilon} \sim N(0, R)$, meaning that the observation error is Gaussian distributed with mean zero and some covariance matrix R.

With the observation in (4.7), we can improve our probability estimate of the state through Bayes theorem, which lets us express the conditional pdf of the state given the observation, as

$$p(\boldsymbol{\psi}^n|\boldsymbol{y}^n) = \frac{p(\boldsymbol{y}^n|\boldsymbol{\psi}^n)p(\boldsymbol{\psi}^n)}{p(\boldsymbol{y}^n)}. \tag{4.8}$$

Here, we recognize $p(\boldsymbol{\psi}^n)$ from (4.4), which we now call the *prior distribution* as this is our pdf of the state *prior* to using the observation. Furthermore, $p(\boldsymbol{y}^n|\boldsymbol{\psi}^n)$ is the *likelihood* and can be interpreted as the probability of observing $\boldsymbol{y}^n$ if we assume that $\boldsymbol{\psi}^n$ is the true state of the system. The denominator $p(\boldsymbol{y}^n)$ is called the *marginal probability* and is the probability that we observe the value $\boldsymbol{y}^n$. However, since $\boldsymbol{y}^n$ is fixed at each observation

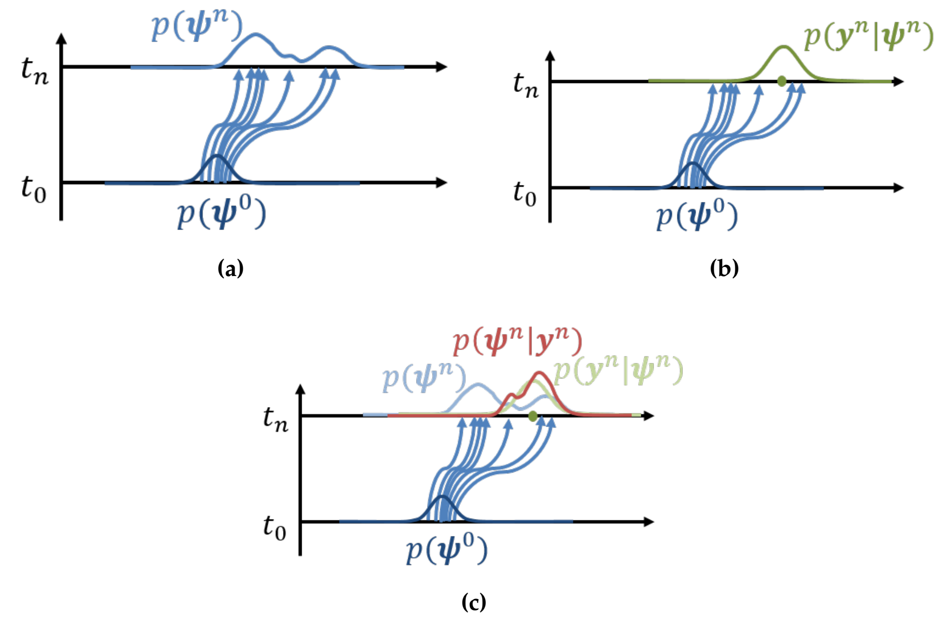


Figure 4.3: Illustration of Bayes theorem. By the use of (a) the prior distribution $p(\boldsymbol{\psi}^n)$ and (b) the likelihood $p(\boldsymbol{y}^n|\boldsymbol{\psi}^n)$, we can find (c) the posterior distribution $p(\boldsymbol{\psi}^n|\boldsymbol{y}^n)$.

time t_n, $p(\boldsymbol{y}^n)$ will be the same for all possible states $\boldsymbol{\psi}^n$, and it is therefore common to consider the marginal probability as a normalization constants.

Figure 4.3 illustrates how Bayes theorem works. In (a) we have the prior distribution $p(\boldsymbol{\psi}^n)$ represented by our ensemble. We then have an imperfect observation $\boldsymbol{y}^n$ in (b), and for each possible state value, we find the likelihood for observing $\boldsymbol{y}^n$ from that state. By combining (multiplying) these two pdfs, we get the posterior distribution in (c), which is a better probability distribution for the true state than $p(\boldsymbol{\psi}^n)$.

Bayes theorem in (4.8) is the backbone of the data assimilation problem. For geophysical applications there are mainly three different methods for solving it:

Ensemble Kalman filters [28, 97]: Solves (4.8) analytically under the assumption of Gaussian errors and linear model and observation operators, and uses the ensemble to represent the uncertainty in the model.

Variational methods [4, 6, 59]: By assuming that the errors in the model and observations are Gaussian, (4.8) can be solved as an optimization problem. This is the most commonly used data assimilation technique in operational weather prediction systems.

Particle filters [72, 95, 94, 97]: Nonlinear data assimilation methods that use a direct ensemble approach for solving (4.8). The filters assign weights to the ensemble members according to their likelihoods, and discards unsuccessful members in exchange for duplication (resampling) of successful ones.

We will now briefly look at the concepts behind ensemble Kalman filters and variational methods as these are the most commonly used data-assimilation methods. We will thereafter devote a longer section on particle filters, as they are the most relevant data assimilation method to this thesis.

Ensemble Kalman filters

The ensemble Kalman filter (EnKF) [28] is perhaps the most widely used ensemble-based data assimilation method in the geosciences. It builds on the Kalman Filter, which solves the data assimilation problem analytically under certain assumptions. As long as we have Gaussian model errors and observation errors, and linear model and observation operators, the posterior distribution in (4.8) will also be Gaussian $N(\boldsymbol{\psi}^{n,a}, P^{n,a})$, with mean and covariance given by

$$\boldsymbol{\psi}^{n,a} = \boldsymbol{\psi}^{n,f} + K\left(\boldsymbol{y}^{n} - H\boldsymbol{\psi}^{n,f}\right) \tag{4.9}$$

and

$$P^{n,a} = (1 - KH)\,P^{n,f}, \tag{4.10}$$

respectively. Here, $\boldsymbol{\psi}^{n,f}$ and $P^{n,f}$ are the mean and covariance, respectively, of the prior distribution, which is often called *the forecast*. The posterior state $\boldsymbol{\psi}^{n,a}$, on the other hand, is called the *analysis*. Furthermore, K is the *Kalman gain*, defined by

$$K = P^{n,f}H^{T}\left(HP^{n,f}H^{T} + R\right)^{-1}. \tag{4.11}$$

The main challenge in the Kalman filter is to define the covariance matrix $P^{n,f}$ for the state vector.

Evensen [27] suggested that the forecast mean and covariance matrix could be estimated through an ensemble. By replacing $\boldsymbol{\psi}^{n,f}$ with

$$\overline{\boldsymbol{\psi}}^{n,f} = \frac{1}{N_e}\sum_{i=1}^{N_e}\boldsymbol{\psi}_i^{n,f}, \tag{4.12}$$

and computing the estimate for the variance as

$$P^{n,f} = \frac{1}{N_e - 1} \sum_{i=1}^{N_e} \left(\boldsymbol{\psi}_i^{n,f} - \overline{\boldsymbol{\psi}}^{n,f} \right) \left(\boldsymbol{\psi}_i^{n,f} - \overline{\boldsymbol{\psi}}^{n,f} \right)^T, \tag{4.13}$$

we can use (4.9) to update each ensemble member individually, and we hence get $\{\boldsymbol{\psi}_i^{n,a}\}_{i=1,\dots,N_e}$. Note that (4.10) is no longer needed, as the covariance for the analysis automatically will be updated through the new ensemble states.

This original version of the EnKF faced two problems in particular, and the first was that it gave a too narrow posterior distribution. It was therefore suggested that an observation error should be sampled independently for each ensemble member and added to $H\boldsymbol{\psi}^{n,f}$ in (4.9), so that the uncertainty in the observation was better represented [14, 40]. This method is now known as the stochastic ensemble Kalman filter. The second problem is that when the dimension of the state vector becomes large, the covariance matrix $P^{n,f}$ becomes enormous since it is of size $N_{\boldsymbol{\psi}} \times N_{\boldsymbol{\psi}}$. Various techniques based on localization [40, 41] and/or square-root transforms [68] of the covariance matrix are used to reduce this storage requirement.

For a full introduction to ensemble Kalman filters, see Evensen [28], or for a recent review of a wide variety of different methods, see Vetra-Carvalho et al. [97].

Variational methods

Variational methods are the data assimilation technique of choice in most operational numerical weather prediction systems (see, e.g., [4, 6], or for an oceanographic application [59]). This is because they traditionally are not ensemble-based, and can be applied to a single simulation run. They solves (4.8) (Bayes theorem) as an optimization problem by assuming that the observation error is Gaussian $\boldsymbol{\epsilon} \sim N(0, R)$, and that the forecasted prior state, $\boldsymbol{\psi}^f$, has a Gaussian background error covariance matrix B. The most powerful method is 4DVar, which is applied to the state's development over a time range that includes several different observations.

With these assumptions, the 4DVar method consists of solving (4.8) by writing out the Gaussian form of the posterior, as

$$p(\boldsymbol{\psi}|\boldsymbol{y}) \propto p(\boldsymbol{y}|\boldsymbol{\psi})p(\boldsymbol{\psi}) = \exp\left(-J(\boldsymbol{\psi})\right), \tag{4.14}$$

with, if we have observations at K different time steps,

$$\begin{aligned}J(\boldsymbol{\psi}) &= \frac{1}{2}(\boldsymbol{\psi} - \boldsymbol{\psi}^f)^T B^{-1}(\boldsymbol{\psi} - \boldsymbol{\psi}^f) \\ &\quad + \frac{1}{2}\sum_{n=0}^{K}\left(\boldsymbol{y}^n - H^n\left(M^n\left(\boldsymbol{\psi}\right)\right)\right)^T R^{-1}\left(\boldsymbol{y}^n - H^n\left(M^n\left(\boldsymbol{\psi}\right)\right)\right),\end{aligned} \tag{4.15}$$

The analysis state $\boldsymbol{\psi}^a$ is here the state that maximizes $p(\boldsymbol{\psi}|\boldsymbol{y})$, which is the state that minimizes $J(\boldsymbol{\psi})$. We therefore find $\boldsymbol{\psi}^a$ as the solution of

$$\begin{aligned}\nabla J(\boldsymbol{\psi}) &= B^{-1}(\boldsymbol{\psi} - \boldsymbol{\psi}^f) \\ &\quad - \sum_{n=0}^{K} M^{n,T} H^{n,T} R^{-1}\left(\boldsymbol{y}^n - H^n\left(M^n\left(\boldsymbol{\psi}\right)\right)\right) = 0.\end{aligned} \tag{4.16}$$

The solution is a state that manages to balance its influence from the forecasted state $\boldsymbol{\psi}^f$ and the observations y^n for $n = 1, ..., K$, while following the development defined by the model.

To solve this, we need the adjoint of the tangent linear model M^T, which makes this method much more complex to implement than, e.g., the EnKF. An alternative is also to use a 3DVar method, in which the optimization problem is defined and solved only for one observation time step at the time. Still, we need to define a background covariance matrix B, and one way to do so is by using a hybrid ensemble Kalman-variational approach for representing B through an ensemble [54].

Even though the variational methods are not used in this thesis, I have chosen to mention them due to their robustness and success in complex operational weather forecasting models.

4.3 Particle filters

Contrary to other data assimilation methods, such as ensemble Kalman filters or variational methods, particle filters (see [72, 94, 95] and references therein) make no assumptions of Gaussian probability distributions or linearity of the model or observation operators. They are therefore often described as fully nonlinear data assimilation methods.

Standard particle filters are ensemble-based techniques that use a direct evaluation of Bayes theorem. Similarly to the ensemble representation of the prior distribution in (4.4), we use the ensemble members to evaluate the likelihood as well. Bayes theorem from (4.8) can then be written as

$$p(\boldsymbol{\psi}^n|\boldsymbol{y}^n) \propto \sum_{i=1}^{N_e} w_i^n \delta(\boldsymbol{\psi}^n - \boldsymbol{\psi}_i^n), \tag{4.17}$$

in which w_i^n are weights given by

$$w_i^n = \frac{p(\boldsymbol{y}^n|\boldsymbol{\psi}_i^n)}{\sum_{j=1}^{N_e} p(\boldsymbol{y}^n|\boldsymbol{\psi}_j^n)}. \tag{4.18}$$

The weight for each ensemble member[1] is the normalized evaluation of its likelihood $p(\boldsymbol{y}^n|\boldsymbol{\psi}_i^n)$ and represents its relative *importance* in the ensemble. As an example, if the observation error in (4.7) is Gaussian $\boldsymbol{\epsilon} \sim N(0, R)$, the weight for ensemble member $\boldsymbol{\psi}_i$ becomes

$$w_i^n \propto \exp\left(-\tfrac{1}{2}\left(\boldsymbol{y}^n - H\left(\boldsymbol{\psi}_i^n\right)\right)^T R^{-1}\left(\boldsymbol{y}^n - H\left(\boldsymbol{\psi}_i^n\right)\right)\right). \tag{4.19}$$

Note that whereas the Monte Carlo forecast in (4.4) represents a straight forward discrete pdf, (4.17) is a weighted discrete pdf.

Figure 4.4 (a)-(b) illustrates this technique, which is called *importance sampling*. In (a), we start with the initial ensemble in which all members are independently sampled within the uncertainty of the initial conditions. At this point, all ensemble members have the same weight $w_i^0 = 1/N_e$. We evolve the ensemble members independently forward in time without changing their weights, just as in a Monte Carlo simulation, which gives us the discrete pdf for $p(\boldsymbol{\psi}^n)$. We then, in (b), use the observation $\boldsymbol{y}^n$ and the observation uncertainty, as shown in green, and evaluate new weights for the ensemble members according to (4.18).

At this point, we have several ensemble members that have weights very close to zero, and if we continue by evolving the ensemble to the next observation, we can expect that the states will spread even more, resulting in even fewer members located close to the next observations. This leads to *filter degeneracy*, which means that the ensemble eventually will have all its weight distributed on very few (or just a single) ensemble members and therefore loose all its statistical information.

Standard particle filters using sequential importance resampling

To avoid the degeneracy, we can apply *sequential importance resampling* (SIR) at each observation time. Since ensemble members with very low weights have a very small contribution to the total pdf, it is a waste of computational resources to keep evolving them. Instead, it would be beneficial to have a larger number of ensemble states in the high-probability areas close to ensemble members with high weights. This can be achieved by resampling the ensemble members based on their weights. Figure 4.4 (c) shows

[1]Note that ensemble members are called *particles* in the particle filter literature.

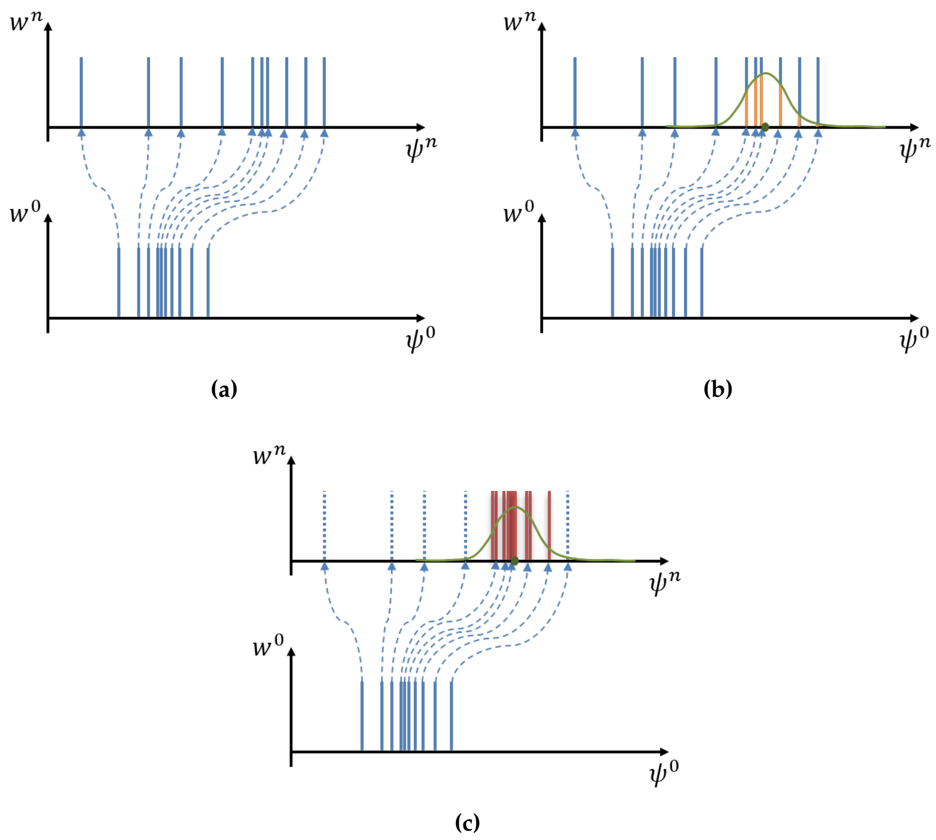


Figure 4.4: Particle filter using sequential importance resampling. We start by (a) evolving an ensemble of equally weighted members forward in time until we have an observation. We then (b) re-evaluate their weights according to the observation and its uncertainty (shown in green), and (c) resample the ensemble based on these weights so that we again get an ensemble of equally weighted members.

such resampling based on the weights in (b). After resampling, we reset the weights back to $w_i^n = 1/N_e$ for all ensemble members and can once again evolve them to the next observation time. Note that if we do not have any stochastic model error (if $\boldsymbol{\beta} = 0$ in (4.3)), we need to perturb the resampled ensemble states to avoid that we have multiple ensemble members that follow the exact same development. Several different schemes exists for doing the resampling, and the interested reader can find an overview in van Leeuwen [94].

The main advantage of the SIR particle filter compared to, e.g., ensemble Kalman filters, is that we never manipulate the state vector of successful ensemble members. This means that the posterior ensemble will consist of states that are consistent with respect to the physics of the model, or slight perturbations of such states.

The great disadvantage, however, is that even with resampling, the particle filters are very prone to collapse when the dimension of the observation N_y is large [89, 90, 94]. When the number of independent observations increases, more ensemble members are likely to end up in the tail of the likelihood, meaning that significant weight will be distributed on fewer particles. The normalized weights will often then be almost one for a single ensemble member, meaning that they are almost zero for all the others. This has the consequence that the posterior again looses all statistical value, and this problem is known as the *curse of dimensionality*.

Particle filters with proposal densities

In recent years, much work has been invested into designing particle filters that avoid the curse of dimensionality (see the recent review paper by van Leeuwen et al. [95]), and a common technique is to use *proposal densities*. The idea is to change the prior distribution, so that instead of evolving the ensemble states as before, we evolve them according to a *proposal distribution* and then compensate for this by altering their weights.

First of all, if we have a Gaussian model error $\boldsymbol{\beta} \sim N(0, Q)$, with Q as the model error covariance matrix, we see from (4.3) that the model step can also be regarded as a random sample from a Gaussian distribution

$$\boldsymbol{\psi}_i^n \sim N\left(M\left(\boldsymbol{\psi}_i^{n-1}\right), Q\right). \tag{4.20}$$

We can therefore consider the prior distribution to be Markovian[2]. Furthermore, if all ensemble members at time t_{n-1} have the same weight, we can write (4.4) as

$$p(\boldsymbol{\psi}^n) = \frac{1}{N_e}\sum_{i=1}^{N_e} p(\boldsymbol{\psi}^n|\boldsymbol{\psi}_i^{n-1}). \tag{4.21}$$

Instead of evolving the ensemble members by sampling them from (4.21), we want to draw them from a proposal density q. The only requirements for q is that it needs to have equal or larger support than $p(\boldsymbol{\psi}^n|\boldsymbol{\psi}_i^{n-1})$, and that it is reasonably simple to sample from it. Besides that, we have large freedom in how we design q. It is therefore possible to choose a proposal density that is conditioned on the observation $\boldsymbol{y}^n$ and all other ensemble states $\boldsymbol{\psi}_{1:N_e}^{n-1}$. Thus, we can, as illustrated by Figure 4.5, use both the

[2]The state $\boldsymbol{\psi}^n$ depends on $\boldsymbol{\psi}^{n-1}$, but not on any previous time steps.

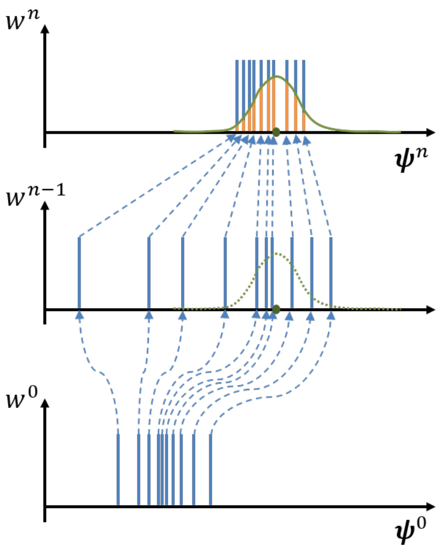


Figure 4.5: Using a proposal density to force the ensemble members to become more similar to the observation during model evolution of the final time step.

observation and the spread in the ensemble to manipulate the ensemble states during the last time step before the observation in such a way that we avoid collapse. Since each ensemble state $\boldsymbol{\psi}_i^{n-1}$ is positioned differently from observation, we can have different proposals for each member, so that the proposal for ensemble member i takes the form of $q_i(\boldsymbol{\psi}^n|\boldsymbol{\psi}_{1:N_e}^{n-1}, \boldsymbol{y}^n)$. By using this proposal, the prior distribution becomes

$$p(\boldsymbol{\psi}^n) = \frac{1}{N_e}\sum_{i=1}^{N_e} \frac{p(\boldsymbol{\psi}^n|\boldsymbol{\psi}_i^{n-1})}{q_i(\boldsymbol{\psi}^n|\boldsymbol{\psi}_{1:N_e}^{n-1}, \boldsymbol{y}^n)} q_i(\boldsymbol{\psi}^n|\boldsymbol{\psi}_{1:N_e}^{n-1}, \boldsymbol{y}^n). \tag{4.22}$$

By using Bayes theorem from (4.8) we get

$$p(\boldsymbol{\psi}^n|\boldsymbol{y}^n) = \frac{1}{N_e}\sum_{i=1}^{N_e} \frac{p(\boldsymbol{y}^n|\boldsymbol{\psi}^n)p(\boldsymbol{\psi}^n|\boldsymbol{\psi}_i^{n-1})}{p(\boldsymbol{y}^n)q_i(\boldsymbol{\psi}^n|\boldsymbol{\psi}_{1:N_e}^{n-1}, \boldsymbol{y}^n)} q_i(\boldsymbol{\psi}^n|\boldsymbol{\psi}_{1:N_e}^{n-1}, \boldsymbol{y}^n). \tag{4.23}$$

Now, by sampling $\boldsymbol{\psi}_i^n \sim q_i(\boldsymbol{\psi}^n|\boldsymbol{\psi}_{1:N_e}^{n-1}, \boldsymbol{y}^n)$, the posterior once again becomes

$$p(\boldsymbol{\psi}^n|\boldsymbol{y}^n) = \sum_{i=1}^{N_e} w_i^n \delta(\boldsymbol{\psi}^n - \boldsymbol{\psi}_i^n), \tag{4.24}$$

but now, the weights are found by

$$w_i^n = \frac{p(\boldsymbol{y}^n|\boldsymbol{\psi}_i^n)p(\boldsymbol{\psi}_i^n|\boldsymbol{\psi}_i^{n-1})}{N_e p(\boldsymbol{y}^n)q_i(\boldsymbol{\psi}_i^n|\boldsymbol{\psi}_{1:N_e}^{n-1}, \boldsymbol{y}^n)}. \tag{4.25}$$

By minimizing the variance among the weights, the *optimal proposal density* is found to be $q_i(\boldsymbol{\psi}^n|\boldsymbol{\psi}^{n-1}_{1:N_e}, \boldsymbol{y}^n) = p(\boldsymbol{\psi}^n|\boldsymbol{\psi}^{n-1}_i, \boldsymbol{y}^n)$ [26]. In the case of a linear observation operator H, and Gaussian model and observation errors given by $\boldsymbol{\beta} \sim N(0, Q)$ and $\boldsymbol{\epsilon} \sim N(0, R)$, respectively, this optimal proposal density is equivalent to $N(\boldsymbol{\psi}^{n,opt}_i, P)$, with

$$\boldsymbol{\psi}^{n,opt}_i = M\left(\boldsymbol{\psi}^{n-1}_i\right) + QH^T\left(HQH^T + R\right)^{-1}\left(\boldsymbol{y}^n - HM\left(\boldsymbol{\psi}^{n-1}_i\right)\right) \tag{4.26}$$

and

$$P = \left(Q^{-1} + H^T R^{-1} H\right)^{-1}. \tag{4.27}$$

Note that $\boldsymbol{\psi}^{n,opt}_i$ has large similarities with $\boldsymbol{\psi}^{a,n}_i$ used by the ensemble Kalman filter in (4.9). The great difference is that we here use the covariance matrix Q for the model error, whereas in the ensemble Kalman filter we use the covariance matrix $P^{n,f}$ for the state itself, which in general is harder to model. Even though this proposal gives a minimal variance among the ensemble weights, it has been shown that it is still not sufficient for avoiding ensemble collapse [89, 90, 1].

A solution to the problem has been suggested in form of the implicit equal-weights particle filter (IEWPF), first by Zhu et al. [101] and later improved with a two-stage update by Skaugvold et al. [88]. The original IEWPF uses a similar but not identical proposal distribution as the implicit particle filter [20], and is the first particle filter that produces uniform weights even in high-dimensional systems. It does, however, have some problems, as it pushes ensemble members further from the observation when the ensemble size is increased, and also under-estimates the spread in the ensemble in low-dimensional numerical experiments. The two-stage scheme alleviates these issues, with the new ensemble members chosen as

$$\boldsymbol{\psi}^n_i = \boldsymbol{\psi}^{n,opt}_i + P^{1/2}\left(\alpha_i^{1/2}\xi_i + \beta^{1/2}\nu_i\right). \tag{4.28}$$

Here, we sample vectors $\xi_i, \nu_i \sim N(0, I)$ in such a way that they are orthogonal, and choose scaling parameters α and β implicitly such that all ensemble members get weights that are equal to the mean of the optimal proposal weights.

The two-stage IEWPF is a central component of Paper III. In that paper, we design a model error covariance structure Q for perturbing the simplified ocean state modelled by the shallow-water equations. Furthermore, our choice of covariance structure enables a massively-parallel implementation of the two-stage IEWPF, so that we can assimilate drifter observations before making forecasts of their drift trajectories.

Chapter 5

Thesis Contribution and Summary of Papers

Efficient forecasting of drift trajectories in the ocean is a highly interdisciplinary problem and the research reported in this thesis builds on four scientific disciplines and topics: numerical mathematics, data assimilation, GPU computing, and oceanography. Several of the papers written during this thesis therefore aim to contribute to more than one of these. Still, all papers are products of all four disciplines, even though this is not explicitly stated in each of them.

The scientific contributions in this thesis are divided on three different levels, as shown in Figure 5.1.

Overview papers: These papers evaluate existing methods for new or more advanced applications.

- **Paper I:** We compare modern finite-volume schemes against traditional finite-difference schemes for oceanographic applications.
- **Paper II:** We show that Python offers a highly suitable programming environment for rapid development of high-performance GPU code, and present results for computational performance and energy efficiency for code written in both CUDA and OpenCL across a wide range of different GPUs.

Proof of concept: This paper presents the suitability of a state-of-the-art method for new and more advanced applications.

- **Paper III:** We use the implicit equal-weights particle filter to assimilate drifter observations into a rotating shallow-water model, and investigate how this improves forecasts of drift trajectories.

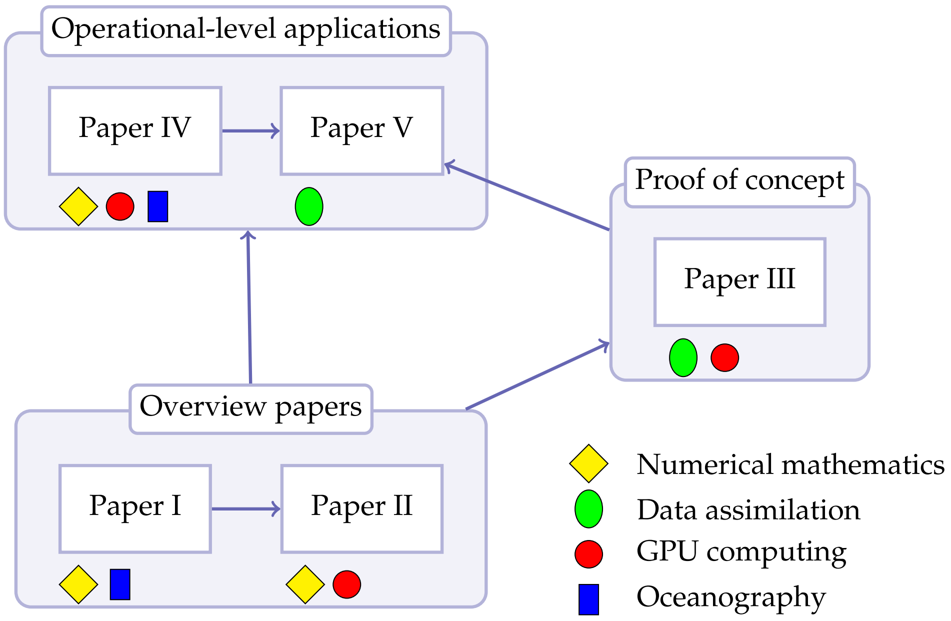


Figure 5.1: An overview of the papers included in this thesis. All four topics are relevant to all papers, but we have highlighted those that are most relevant for each of them. The arrows indicate how the results from some papers are used to realize the research in others.

Operational-level applications: These papers take modern methods out of idealized settings and extend them to tackle challenges found in operational settings.

Paper IV: We extend a modern finite-volume scheme to enable oceanographic simulations of real-world domains through initialization from operational ocean forecasts.

Paper V: We investigate the use of very large ensembles of simplified ocean models for applying a SIR particle filter to forecast drift trajectories.

As indicated by the arrows in Figure 5.1, some of the papers build on the results from others. The overview papers (Paper I and Paper II) have been used to review suitable numerical methods and GPU computing technologies that are further used in the three other papers, whereas the Paper V applies the full simulation framework presented in Paper IV and some components from the data assimilation code and the model error developed during the work for Paper III.

Code releases and reproducible science

Reproducibility is one of the fundamental aspects of the scientific method, as scientific results only have value if they can be replicated independently by others. In this thesis, all results rely on computer code, and simulation frameworks and numerical experiments can rarely be described in sufficient detail in a research paper to enable others to replicate the results purely from the written information (see, e.g., [99]). Releases and publications of the software that produces the presented results are therefore an important contribution from this thesis. In addition to the simulation frameworks, each release also includes Jupyter notebooks used for pre- and post-processing of experiments, so that all figures easily can be recreated. Furthermore, all software is released under an open source license, the GNU Public Licence version 3 (GPLv3), and the latest version of the complete code base can be found at `https://github.com/metno/gpu-ocean`.

In the following pages, we give individual summaries of each paper, followed by a short discussion of the work and description of the corresponding software.

Paper I

Evaluation of Selected Finite-Difference and Finite-Volume Approaches to Rotational Shallow-Water Flow

Håvard Heitlo Holm, André Rigland Brodtkorb, Göran Broström, Kai H. Christensen, Martin Lilleeng Sætra

Communications in Computational Physics, volume 27, number 4, pages 1234–1274, 2020
DOI: 10.4208/cicp.OA-2019-0033

The aim of this work was to investigate different numerical schemes that can serve as a simplified ocean model. The central question for this paper (although not explicitly stated in the paper) was whether the modern high-resolution, finite-volume scheme suggested by Chertock et al. [18], here abbreviated CDKLM, could match the classical, leapfrog, centered-in-time, centered-in-space (CTCS) finite-difference schemes, which has commonly been used in the oceanographic communities. Furthermore, we included the finite-volume scheme by Kurganov and Petrova (KP) [49], which we modified with a naïve discretization of the Coriolis forces to make it applicable to rotational flow. As mentioned in Section 2.3, the main difference between the KP and CDKLM schemes is that the CDKLM scheme is well-balanced with respect to the geostrophic balance, whereas the KP scheme is well-balanced with respect to lake-at-rest. Finally, we also included the finite-difference, forward-backward linear (FBL) scheme [87] in the comparison. Even though this scheme only solves the *linearized* shallow-water equations, it is interesting because of its simplicity and potential for very efficient simulations.

The comparison was made by running all four schemes on a set of seven test cases designed to capture relevant oceanographic physics for problems such as storm surge and drift trajectory modelling. This includes:

Rossby adjustment: Disturbance in the sea-surface that adjusts to a non-trivial rotating steady-state solution.

Kelvin waves: Fast barotropic waves (such as the tides) that travel along the coast line.

Rossby waves: Slow barotropic waves that are caused by spatial variations in potential vorticity, either from a sloping bathymetry (resulting in topographic Rossby waves) or a Coriolis beta-model (planetary Rossby waves).

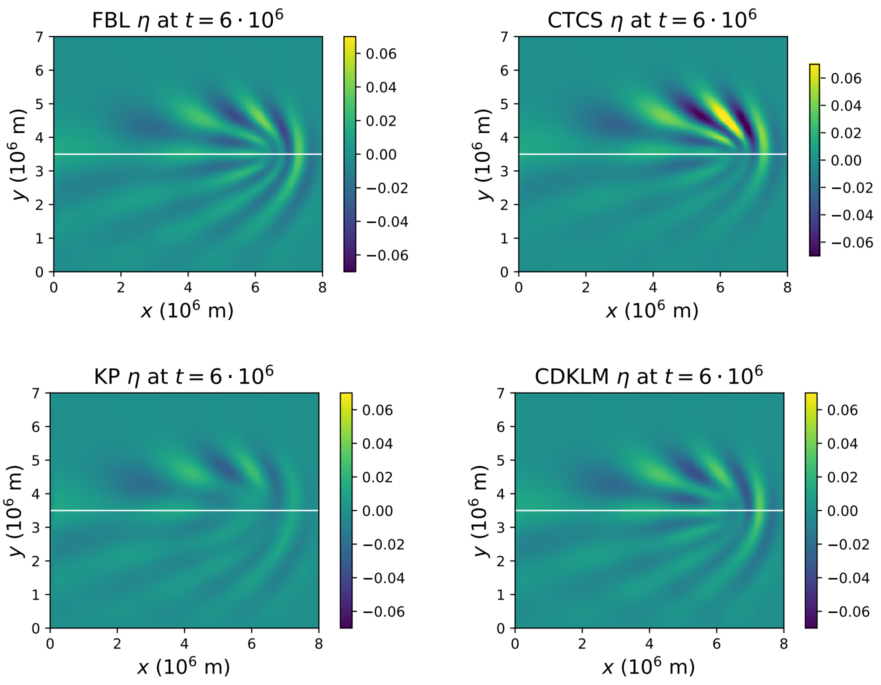


Figure 5.2: Planetary Rossby waves for the four different numerical schemes, driven by a linear Coriolis force along the y-axis. The case is initialized by zero momentum and a bump in the central right part of the domain.

Figure 5.2 shows the sea-surface elevation after simulation of planetary Rossby waves. The test case is initialized with a bump east in the domain and results in Rossby waves propagating from east to west, while wave energy slowly propagates eastwards. Furthermore, we include a test that demonstrates the primary advantage of the finite-volume schemes, which is capturing discontinuous solutions. These results are well-known, as the finite-difference schemes are based on a formulation of the shallow-water equations that are only conservative for smooth solutions and therefore unable to capture shocks. We also compare computational performance and numerical convergence.

Our results show that the CTCS scheme performs well in cases dominated by geostrophic balances, such as Rossby adjustment and Rossby waves, but we were not able to show correct dispersion for large gravity-driven waves. Furthermore, the KP scheme is too diffusive due to its naïve discretization of the Coriolis forces, and the well-balanced reconstruction of CDKLM makes it perform better than KP on all tests. The conclusion is that CDKLM is the superior scheme for gravity-driven motion, such as Kelvin waves, but still has some minor potential for even better treatment of rotational motion compared to CTCS. As expected, we see that the FBL scheme manages to capture all linearized physics and runs one order of magnitude faster than CDKLM.

Comments

The results from this paper made us confident to proceed with our attempt to use modern high-resolution finite-volume methods, here through the CDKLM scheme, in the role of a simplified ocean model that can eventually be used to forecast drift trajectories. We see, however, that it is important to use methods that are well-balanced with respect to the rotational physics, as we observe that the KP scheme is diffusive and looses some of the rotational energy.

Most surprising among the results from this paper, however, is that the CTCS scheme demonstrates a slow-down during simulations of Kelvin waves, even though the theory states that the wave crest should be slightly faster than the trough. We spent significant time debugging the CTCS scheme and discussed these counter-intuitive results with Lars Petter Røed, who has worked extensively with such schemes. Together, we found an explanation to this behavior, which is outlined in Section 4.4 in the paper. After the paper got accepted, Røed came back to us after implementing the Kelvin waves test case himself, and his new results showed that the crest gained speed, which is contrary to our results but more in accordance with theory. The consequence of the expected behavior, however, is that a shock builds up in the front of the wave, causing CTCS to break down, since it is unable to simulate discontinuities. This means that even with these new results, our conclusion of proceeding with the CDKLM scheme is unaltered.

A similar study has been carried out by Pankratz et al. [66]. They compared a fourth order high-resolution finite-volume scheme against a first and a second order "oceanographic" finite-difference scheme, for simulations of large eddies and barotropic jets. In their conclusions, they highlight that the finite-volume scheme is much more stable but also more expensive compared to the finite-difference schemes. Overall they show, as we do, that modern high-resolution finite-volume schemes can be used for oceanographic applications.

Since the main aim of this work was to evaluate the suitability of different numerical schemes in the role of simplified ocean models, the paper focus mostly on the numerical results without going into a deep mathematical analysis of the chosen schemes. A more thorough analysis of geostrophic adjustment through a comparison of the exact and discrete dispersion laws (see, e.g., [5, 15]) could therefore be considered in a future work.

Software

We have released the software as supplementary material, registered with the DOI 10.5281/zenodo.3204200. This release contains the source code with the implementations of the four numerical schemes and Jupyter notebooks for running the test cases and plotting the results.

Paper II

GPU Computing with Python: Performance, Energy Efficiency and Usability

Håvard Heitlo Holm, André Rigland Brodtkorb, Martin Lilleeng Sætra

Computation, volume 8, number 1:4, 2020
(Special issue on Energy-Efficient Computing on Parallel Architectures)
DOI: 10.3390/computation8010004

High-performance computing (HPC) applications in general, and GPU-accelerated code in particular, are traditionally implemented in relatively low-level programming languages, such as C/C++ or Fortran. The most important thing for performance, however, is that the main computational engine of the application is implemented to run as efficiently as possible (see discussion in Section 3.3). The topic of this paper is therefore to evaluate the usability of accessing the GPU through Python. We also evaluate the performance and energy efficiency of such applications using Python packages for the two most common GPU programming models (PyCUDA vs PyOpenCL [45]), across different GPUs (HPC GPUs, commodity-level gaming GPUs, and laptop GPUs), and for different levels of code optimization (first implementations vs optimized code). To evaluate these aspects, we use the three most promising numerical schemes from Paper I: FBL, CTCS, and CDKLM, which in the paper are described as the linear, nonlinear, and high-resolution schemes, respectively.

Our findings show that any reductions in computational efficiency induced by using Python instead of C++ for accessing CUDA and OpenCL kernels is negligible. Moreover, we find that applications written in CUDA and OpenCL, and tuned to an equivalent optimization level, most often show the same computational performance on the same GPU. We further observe that the effect of doing profile-driven code optimization varies substantially between different GPUs. These last two findings are elaborated in Figure 5.3, which shows relative computational performance for four different versions of the three numerical schemes across seven different GPUs. Similarly, we also found that the energy consumption did not depend on whether we use CUDA or PyOpenCL, but rather that computationally efficient code in general seems to also be energy-efficient code. Optimal energy efficiency, however, is not obtained at optimal computational efficiency, and therefore requires specific tuning of the code and GPU parameters.

Finally, the paper evaluates the usability of PyCUDA and PyOpenCL for rapid development of GPU code through Python. Whereas both Python

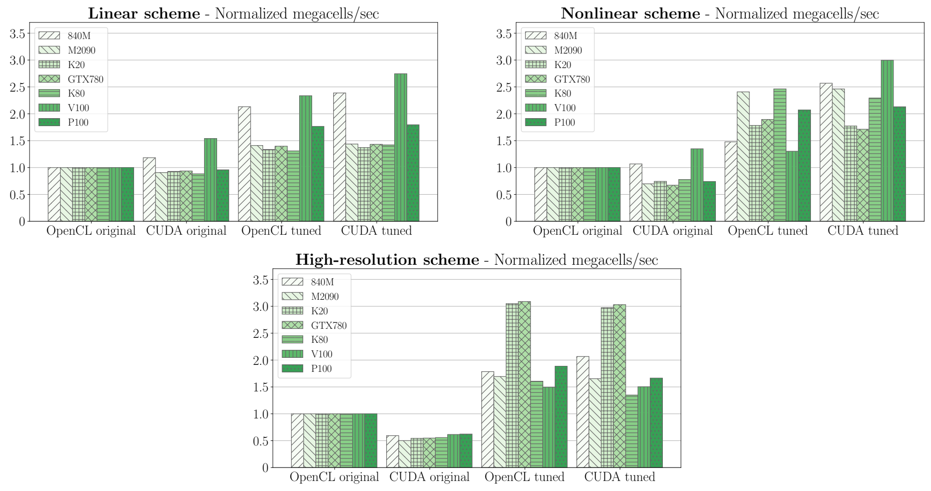


Figure 5.3: Computational performance for the three different numerical schemes on seven different GPUs. Each figure shows benchmark results for the original and optimized versions of each scheme in both CUDA and OpenCL, showing that the effect of optimization varies a lot among GPUs, but is more or less invariant to the choice of using CUDA or OpenCL.

packages cover the complete underlying APIs of CUDA and OpenCL, the CUDA platform has a more extensive ecosystem compared to OpenCL. For instance, CUDA offers better tools for profiling and a more integrated selection of libraries through its software development kit (the CUDA SDK), whereas PyOpenCL relies on third-party libraries. We consider this to be of grater value than the opportunity to run the code on all types of GPUs and CPUs, which is the main advantage of OpenCL over CUDA.

Comments

This paper is an extended version of a conference paper [38] that has been accepted for publications in the *Proceedings of the International Conference on Parallel Computing, ParCo2019*, and that was presented in the mini-symposium on “Energy-efficient Computing on Parallel Architectures (ECO-PAR)” during the conference. Afterwards, we were invited to contribute an extended version (this paper) to an upcoming special issue journal on *Energy-Efficient Computing on Parallel Architectures*.

Even though GPUs are well known to have great potential in energy-efficient computing, we had no prior experience working with this topic.

In general, HPC facilities have a very large power consumption. The most powerful supercomputer on the November 2019 Top 500 list [58] uses the same amount of energy in 75 minutes alone as an average household in Oslo consumed during the year 2012 [91]. An increased focus on energy-efficient computing could therefore be both prudent and timely. Contributing to this mini-symposium and the up-coming journal special issue has therefore been an interesting additional aspect to the main topic of this thesis.

Two of the results from this paper have had a direct impact on the subsequent work in this thesis. First and foremost, the experiences from working in PyOpenCL made us change our choice of GPU computing platform from PyOpenCL to PyCUDA. Even though our GPU programming experience was mainly from working in CUDA, we wanted to develop the code for this project in OpenCL, which is a more open platform that supports execution on any GPU and even on CPUs. It is, however, hard to profile OpenCL applications, as profiling in OpenCL requires that we explicitly add hardware counters to the source code instead of simply running the original program through a profiler, such as Nvidia Visual Profiler, which is available for CUDA. The advantage of running the code on multiple platforms turned out to be of less importance, as we did not have access to AMD hardware, even though we knew that including a non-Nvidia GPU in our experiments would improve the paper significantly. By taking the cost of porting our initial simulation code from (Py)OpenCL to (Py)CUDA, we were rewarded by working in a richer software environment. The second impact from writing this paper was that we got significantly faster code through the profile-driven code optimization carried out for the experiments. This improved all aspects of the later papers, both in the context of rapid prototyping and through making it feasible to run larger ensembles for Papers III and V.

Software

The experiments presented in this paper evaluate multiple versions and optimization levels of the same software. The scripts running these experiments therefore rely on cloning and checking out all of these versions from the git repository it is assumed to be ran from. For this reason, a software release outside of the git repository is of limited value for reproducing this work, and we decided not to release it in a permanent repository.

The code can still be accessed from our Github repository at `https://github.com/metno/gpu-ocean`. All experiments in the paper can be run from the dedicated branch named `paper/energy-efficiency`.

Paper III

Massively Parallel Implicit-Equal Weights Particle Filter for Ocean Drift Trajectory Forecasting

Håvard Heitlo Holm, Martin Lilleeng Sætra, Peter Jan van Leeuwen

Journal of Computational Physics: X, volume 6, number 100053, 2020.
DOI: 10.1016/j.jcpx.2020.100053

The paper targets the main topic of this thesis by investigating nonlinear data assimilation of drifter and mooring observations in a shallow-water model for forecasting drift trajectories. We present a massively-parallel GPU implementation of the implicit equal-weight particle filter (IEWPF) and demonstrate our simulation system on a challenging test case containing near-realistic chaotic instabilities. This is the first massively parallel implementation ever of such a particle filter, and also the first time that IEWPF has been applied to a high-dimensional geophysical application.

The key element for enabling an efficient parallel implementation of IEWPF is to define a local covariance structure Q to the stochastic model error. We therefore present an embarrassingly parallel algorithm for applying such a covariance structure to normal distributed random numbers. Our choice of Q is made to adhere to the geostrophic balance and has a local computational structure that is also flexible with respect to correlation radius. The paper contains detailed descriptions of all stages of the IEWPF algorithm, and we demonstrate how the choice for Q has an impact in every algorithmic step.

The paper presents numerical experiments using an ensemble of simplified ocean states to capture near-realistic currents. The experiments are initialized from an unstable steady state consisting of two geostrophic jets, in which a minimal disturbance of the state leads to a chaotic instability. This means that ocean states that are perturbed in slightly different ways will have very different developments. A Monte Carlo simulation therefore results in a smooth ensemble mean, as shown at the top of Figure 5.4. We then use a single simulation as the true state, from which we make observations from 240 anchored moorings in a grid pattern and from 64 free drifters. Since the truth is simulated using the same model as we use to make forecasts, these experiments are so-called identical twin experiments. The middle and bottom rows of Figure 5.4 show the ensemble means after assimilating observations from the drifters and moorings, respectively. The drifter observations help the ensemble capture some of the main curves in the two jets, whereas the observations from the moorings lead to an ensemble mean with very fine details. Even though the mooring experiment

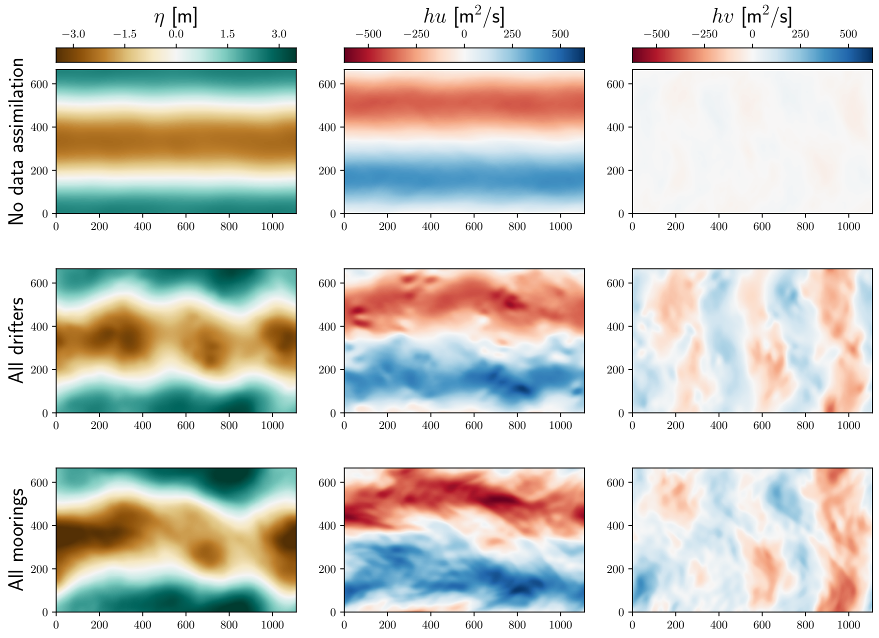


Figure 5.4: Ensemble mean of the sea-surface level (η), jet flow (hu), and cross-jet flow (hv) after no data assimilation (top), assimilation of drifter observations (middle row), and observation of mooring data (bottom). The axes are in km.

uses observations of approximately 0.1% of the state space, the resulting ensemble mean is almost indistinguishable from our synthetic true state.

Based on the results in Figure 5.4, we proceed by making ensemble forecasts of drift trajectories, as shown in Figure 5.5. The Monte Carlo experiment gives a large spread from the very beginning, whereas observations from the drifters show reasonable forecasts with reduced uncertainty during the first 12–24 hours. The forecast produced after assimilating observations from the moorings, however, is exceptionally good for the entire forecast period up to three days.

Additionally, we find that the forecast quality varies significantly when using observations from only half of the domain, depending on whether we observe one jet completely or two jets partially. We also show indications of good statistical quality of our results, based on rank histograms resembling uniform distributions. As expected, we confirm that the standard particle filter leads to ensemble collapse for the same ensemble sizes as used in the numerical experiments. Finally, we include an analysis of computational performance and bottlenecks of the experimental setup.

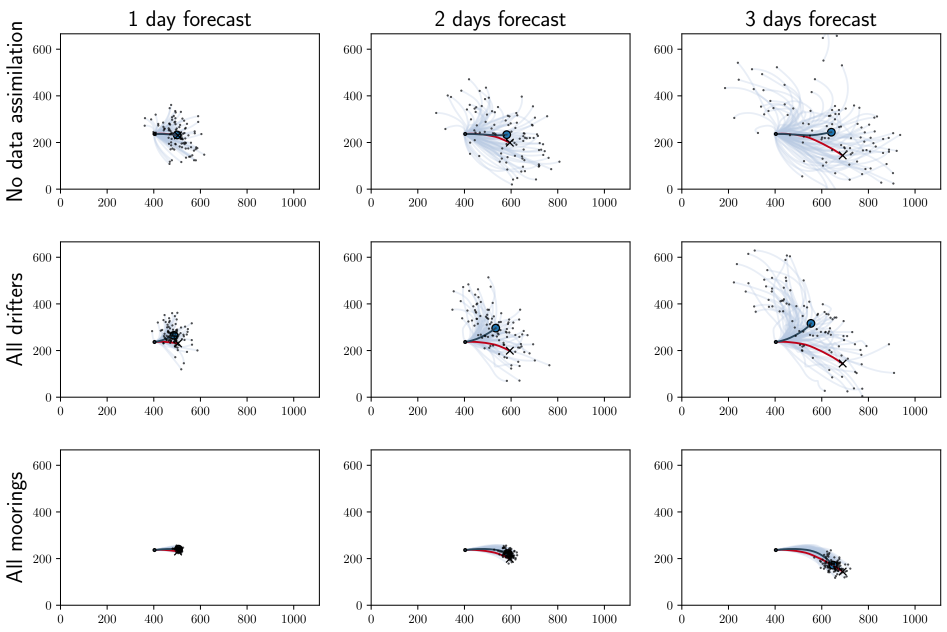


Figure 5.5: Ensemble forecast of drift trajectories starting from the states in Figure 5.4, shown after one, two and three days. Light blue lines are the ensemble trajectories, dark blue is the ensemble mean, and red is the truth.

Comments

Even though we demonstrate a successful application of IEWPF on complex simulations using a shallow-water model, there is still room for improvement. One of the weaknesses of IEWPF seems to be its dependency on the covariance structure Q of the model error. With our choice of Q, the particle filter corrects the ocean state by adding a dipole structure (a positive and negative bump rotating in opposite directions) centered on the observation. This will assimilate the observation into the state at the correct location, but seems sometimes to give a worse representation of the true state in the surrounding area. We believe this might be a reason for why the assimilation of drifter observations has relatively little effect on the true state, as the location of the observations changes all the time and thereby causing side effects to harm recently corrected grid cell values.

Another interesting future work would be to account for a varying bathymetry and land mask to enable experiments in fully realistic domains, such as those considered in Paper IV and Paper V. This could be achieved by modifying and improving the IEWPF algorithm presented in the paper, or by finding a way to loosen up this requirement. Some alternatives for the latter option are mentioned in the paper, such as using a reduced gravity model or a multilayered system.

A missing, but interesting aspect would be to extend the numerical experiments by comparing the IEWPF against a suitable ensemble Kalman filter (EnKF), with regards to both forecast quality and computational requirements. Since EnKF uses covariance structures found in the ensemble states, we will avoid the problem of choosing a single covariance structure. On the other hand, we can expect a negative impact on computational performance because of the ensemble-wide data dependency for computing these covariance structures.

Software and data

The source code for the software and experiments described in this paper is released as supplementary material, registered under the DOI 10.5281/zenodo.3458291.

Replication of the experiments relies on access to the synthetic true state and the initial states for the ensemble members. Our raw simulation results are also of interest, as we show trajectory forecasts for only a selection of drifters in the paper, and other post-processing methods might reveal new interesting interpretations of the results. All these datasets are therefore released as supplementary material as well, registered under the DOI 10.5281/zenodo.3457538.

Paper IV

Real-World Oceanographic Simulations on the GPU using a Two-Dimensional Finite Volume Scheme

André Rigland Brodtkorb, Håvard Heitlo Holm
Preprint: arXiv:1912.02457, 2019
In review

In this paper, we present a simulation framework for solving the rotational shallow-water equations based on a modern high-resolution finite-volume method (the CDKLM scheme). The framework can be used directly with real-world data from operational ocean forecasts to run lightweight, GPU-accelerated ocean simulations. The code is based on the CDKLM simulator, which was first implemented for Paper I and later optimized in Paper II. In this work, we extend the scheme with several features required for running real-world ocean simulations. The extensions include spatially varying north vector and Coriolis term, moving wet-dry interface, a static zero-flux landmask, bottom friction, wind forcing, boundary conditions for nesting the simulation into a larger model, and an efficient reformulation that makes the scheme well-suited for execution on GPUs.

We demonstrate the framework through three numerical experiments on different sections along the Norwegian coast: first, a section of the Norwegian Sea completely without interaction with land; then, an area around the Lofoten islands with a very complex land mask (see Figure 5.6); and finally, we simulation the complete coast of Norway. All experiments use initial, boundary, and forcing data, along with bathymetry and landmask, from operational ocean forecasts issued by the Norwegian Meteorological Institute. These forecasts are generated through a three-dimensional ROMS model, but the forecast files are still distributed with an additional vertically-integrated momentum field.

Each of the three experiments are run with original, double, and half resolution compared to the reference forecast. The results show that the high-resolution simulations manage to maintain more of the fine structures that are present in the reference solutions, whereas the low-resolution simulations smear out the details and are dominated by grid effects. We also report tidal forecasts for selected locations along the Norwegian coast, which are general in good agreement with the reference forecast, and show that even with several of the new extensions, we obtain the expected numerical convergence.

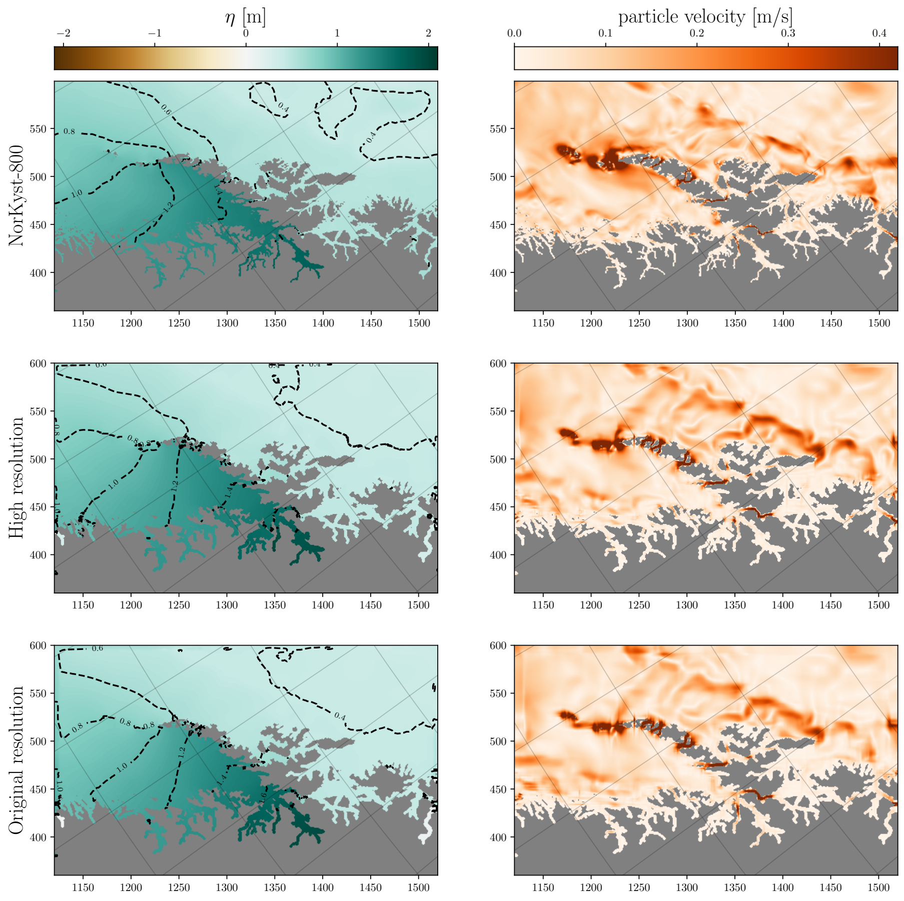


Figure 5.6: Comparison between the NorKyst-800 reference solution (top row) and our simulation framework running with double resolution (mid row) and the same resolution (bottom row) as NorKyst. The left column shows sea level η, whereas the right column shows particle velocity. The domain shows the area surrounding the Lofoten islands in Northern Norway (see also Figure 8 in Paper IV) with the lines representing latitude and longitude. North points up towards the right.

Comments

Many numerical methods are often presented and demonstrated on relatively simple experiments. This paper therefore aims to lift the CDKLM scheme towards an operational level, describing all additional complexities encountered on the way. Furthermore, the main motivation behind the paper is to explore how well the shallow-water equations and the CDKLM scheme are able to capture short-term oceanographic physics when applied to real-world data. This paper is therefore also central to the main topic of this thesis.

One interesting question arising from this study, is what happens to features that are built up and driven by baroclinic instabilities. Recall that barocline dynamics are driven by pressure gradients due to variations in temperature and salinity, which are not modelled by the shallow-water equations. As can be seen in Figure 5.6, the reference solution contains a string of eddies crossing diagonally through the domain from the top and towards the right (in east-northeast direction). Our high- and normal resolution results still manage to preserve a similar structure after 24 hours of simulation, and it could therefore be interesting to study more closely what happens to these structures during the first simulated hour, and how representative our solution is to the actual ocean currents.

It should be noted that for large domains, e.g., the complete domain of the operational ocean forecast, the use of Cartesian grids introduce an error as the curvature of the Earth is not taken into account. A viable technique that avoids this error is to discretize the shallow-water equations on a spherical grid based on lat-lon coordinates [16]. Our main motivation, however, is to investigate in methods suitable for short-term predictions of drift trajectories. As these are local problems, we aim to use this framework on domains that are smaller than those shown in the the examples of this paper, and for which Cartesian discretizations are sufficiently accurate.

The reformulation of the original CDKLM scheme to make it suitable for GPU implementation, was developed early during this Ph.D. project, and has already been used in Papers I – III. However, since this reformulation has not been directly relevant to the main topics of those papers, it was not published until in this paper.

Software

The source code for the framework presented in this paper is currently available in revision `7281301a3a286351b6e857add12c2fbd0165b229` of the `gpu-ocean` repository on Github, `https://github.com/metno/gpu-ocean`. We plan to release this software on Zenodo to get a DOI, similarly to Papers I, III, and V, pending possible acceptance of the paper.

Paper V

Data Assimilation for Ocean Drift Trajectories Using Massive Ensembles and GPUs

Håvard Heitlo Holm, Martin Lilleeng Sætra, André Rigland Brodtkorb

Accepted for publication in the proceedings of the conference Finite Volumes for Complex Applications IX, 2020

This final paper presents forecasts of drift trajectories after using fully nonlinear data assimilation on massive ensembles of simplified ocean models. We base our ocean models on the framework presented in Paper IV, and use mpi4py, a Python package for parallelization using the MPI message passing interface, to run a large ensemble across multiple processes and GPUs. We demonstrate this extended framework by running ensembles with up to 10 000 ensemble members on a Nvidia DGX-2 server that contains 16 Nvidia V100 GPUs. Since we can run ensembles this large, we now use a particle filter based on sequential importance resampling (SIR), but to avoid the curse of dimensionality, we are still restricted to a low number of observations.

Contrary to in Paper III, we do not base the observations on an identical twin simulation, but rather create synthetic drift data using the operational drift trajectory simulator OpenDrift [23] with operational ocean forecasts. Even though we configure OpenDrift to only use the vertically integrated ocean currents, the *truth* is generated by a different and more complex physical model compared to the one used for our ensemble. Figure 5.7 shows an overview of the experimental setup presented in this paper, along with an illustration of the SIR particle filter.

We present three different numerical experiments, in which we first run 48 hours of data assimilation, followed by 24 hours of trajectory forecast for four drifters. The first is a pure Monte Carlo simulation, in which we do not use any observations. The second and third experiments assimilate observations from the four drifters every 30 and 5 minutes, respectively. We find that the use of observations improves the quality of the trajectory forecast, and observations every five minutes leads to a better forecast than observations every 30 minutes. We do, however, also observe that the ensemble struggles to capture the true drift trajectories in areas dominated by barocline dynamics, even with short data assimilation windows.

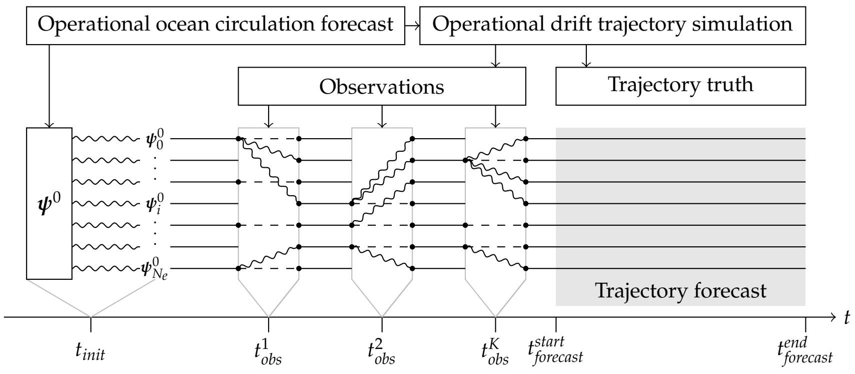


Figure 5.7: Algorithmic overview of the simulation, data assimilation and drift trajectory forecast. Straight lines are deterministic simulation, wiggly lines are perturbations, and dashed lines show ensemble members that are kept during the resampling phase.

Comments

This paper combines all the aspects of this thesis and demonstrates a proof-of-concept for the use of a massive ensemble of GPU-accelerated simplified ocean models along with nonlinear data assimilation for prediction of drift trajectories. Based on this conference paper, we see several possibilities for extending this work with a more detailed description of the parallel algorithms and data assimilation framework, as well as running more experiments to facilitate for a deeper discussion concerning the results.

First, it would be interesting to run a large number of experiments using observations from an increasing number of drifters to look at the resampling statistics. Snyder et al. [89] show that in order to avoid collapse, the ensemble size needs to scale exponentially with the number of drifters. This means that even with 10 000 ensemble members, we should still not expect to be able to use a large number of drifters for observations. Another interesting aspect would be to explore whether our framework works better in certain areas compared to others. Could we, for instance, make more accurate forecasts if we run a similar experiment in the Barents sea, which is dominated by barotropic flow and therefore, in theory, better captured by our model? A third aspect that could be improved is the parallel execution of ensemble members residing on the same GPU. This way, we can increase the utilization of the GPU resources and achieve even faster simulations. We also know that we could decrease communication costs by using direct GPU-to-GPU communication for GPUs that are located on the same compute node, but this is a less pressing issue, since communication is not a bottleneck in the current experiments. Furthermore, since SIR particle filters make no assumptions on the probability distributions in the problem, we are free to explore any covariance structure for the model error that we want. We can even apply different model error distributions to different ensemble members, and let the particle filter resample those structures that best fit with the observations.

Finally, we note that the implementation strategy we have used for running an ensemble on multiple nodes with MPI is highly flexible. We therefore expect no major challenges in also using the same parallel framework for data assimilation experiments using IEWPF on multiple GPUs.

Software

Full source code for the simulation code and numerical experiments described in this paper is released as supplementary material, registered under the DOI 10.5281/zenodo.3591850.

Chapter 6

Summary and Outlook

This thesis has focused on topics related to short-term forecasting of drift trajectories in the ocean. Today, the Norwegian Meteorological Institute issues such forecasts operationally by the use of complex three-dimensional ocean models and dedicated drift trajectory software. We have here investigated a complementary approach, based on the following aspects of the problem. First of all, the large uncertainties in initial conditions make real-world ocean forecasts a challenging problem. Instead of running a single complex and computationally intensive simulation, we have investigated using a large ensemble of simplified ocean models, thereby hoping to trade some of the model accuracy into better uncertainty quantification. Second, we have used data assimilation to improve the probabilistic forecasting through assimilating observations of the ocean into such an ensemble. Since the model is highly nonlinear and there are few observations, particle filters arise as good candidates. Finally, it is challenging to initialize ensembles of three-dimensional ocean models, as they are hard to perturb in a physically consistent way. Simplified models based on the shallow-water equations are, on the other hand, easier to perturb, which enables us to more easily initialize such ensembles.

The most central contributions in this thesis are:

- Assessment of the applicability of a modern, high-resolution finite-volume scheme for solving the rotating shallow-water equations to oceanographic problems, and development of a framework based on this scheme for simulations using real-world data from operational ocean forecasts.
- Massively parallel implementation of a recent state-of-the-art particle filter, assimilating drifter observations into an ensemble of simplified ocean models.

- Evaluation of performance and energy efficiency of equivalent versions of untuned and optimized numerical schemes in CUDA and OpenCL across seven different GPUs, and a usability study of the Python programming environment for efficient development of high-performance GPU code.

Future work

As this thesis is highly interdisciplinary, it is possible to build on this work in multiple directions. The most immediate and relevant directions for improving our current framework for drift trajectory forecasting, however, seem to be through improving the model by introducing a two-layer model, exploring alternative model error distributions, and/or testing other data assimilation methods.

Extending the simplified ocean model

In this work we have used a purely barotropic simplified ocean model based on vertically-averaged ocean currents, and the numerical experiments in Paper IV show that this works well for tidal forecasts. When forecasting real-world drift in the uppermost layers of the ocean, however, this simplification is not necessarily very accurate, as currents in this layer often are stronger than the vertical mean. Additionally, neglecting baroclinic processes can also lead to inaccuracies in the short-term predictions, as we see from some of the experiments for drift forecast in Paper V.

One way of improving both these factors is by using a two-layer shallow-water model (see, e.g., [76]). Such a model consists of two shallow-water equations stacked on top of each other, with the lower one having a modified pressure term and slightly higher density than the upper one, while also acting as a time-dependent bathymetry for the upper layer. The layers also interact through shear stress at their interfaces. This model can simulate baroclinic responses if the thickness of the lower layer increases while the thickness of the upper layer decreases, and might lead to more realistic modelling of the relevant ocean currents for drift modelling.

Another alternative to the two-layer model is also the 1.5-layer model, in which we assume that we have a relatively thin upper layer on top of a thick lower layer. In this model, the motion of the lower layer is neglected, and we use a reduced gravitational force to account for the pressure differences between the two layers. This model is therefore very similar to the regular shallow-water equations, and a low-hanging fruit for future work would be to compare this model with our current approach for forecasting drift trajectories.

Model error covariance structures

As discussed in the summary of Paper III in Chapter 5, the assimilation of observations through the IEWPF algorithm depends strongly on the covariance structure for the model errors. Further research to find improved or more realistic covariance structures might therefore improve how IEWPF assimilates observations into the ensemble states.

A similar direction is to use SIR particle filters to explore different forms of model perturbations. Since the data assimilation step in SIR is independent of the model errors used, the framework developed for Paper V can serve as a flexible testing ground for such work. Some ideas for possible perturbations are to change the tidal signal through the boundary conditions, use different wind forcing, or perturb the bathymetry. An initial idea from the early stages of the project was to perturb the locations of ocean eddies, as they have a large impact on drift trajectories, but more research is needed to come up with a good strategy for achieving this in practise.

An alternative could also be to experiment with the use of machine-learning techniques to learn the missing physics in the simplified ocean model [98]. As training data, we could possibly run large numbers of simulations using our shallow-water model and compare the results with corresponding operational ocean forecasts.

Alternative data assimilation methods

In this work, we have applied two different particle filters for assimilating observations from drifters: the traditional particle filter using SIR, and the recently proposed IEWPF. The search for new and better ways to avoid the curse of dimensionality is, however, an active research topic with several proposals for improved particle filters every year, as illustrated by the recent review paper by van Leeuwen et al. [95]. It would therefore be interesting to try other particle-filter techniques on this problem as well, such as, e.g., hybrid methods between ensemble Kalman filters and particle filters [75], for comparison against IEWPF.

Bibliography

[1] M. Ades and P. van Leeuwen. An exploration of the equivalent weights particle filter. *Quarterly Journal of the Royal Meteorological Society*, 139(672): 820–840, 2013. doi: 10.1002/qj.1995.

[2] J. Albretsen, A. Sperrevik, A. Staalstrøm, A. Sandvik, and F. Vikebø. NorKyst-800 report no. 1: User manual and technical descriptions. Technical Report 2, Fisken og Havet, Institute of Marine Research, 2011.

[3] R. Bleck. An oceanic general circulation model framed in hybrid isopycnic-Cartesian coordinates. *Ocean Modelling*, 4(1):55 – 88, 2002. ISSN 1463-5003. doi: 10.1016/S1463-5003(01)00012-9.

[4] M. Bonavita, Y. Trémolet, E. Holm, S. Lang, M. Chrust, M. Janiskova, P. Lopez, P. Laloyaux, P. de Rosnay, M. Fisher, M. Hamrud, and S. English. A strategy for data assimilation. *ECMWF Technical Memoranda*, (800), 2017. doi: 10.21957/tx1epjd2p.

[5] F. Bouchut., J. Le Sommer, and V. Zeitlin. Frontal geostrophic adjustment and nonlinear wave phenomena in one-dimensional rotating shallow water. Part 2: High-resolution numerical simulations. *Journal of Fluid Mechanics*, 514:35—63, 2004. doi: 10.1017/S0022112004009991.

[6] F. Bouttier and P. Courtier. Data assimilation concepts and methods. 2002. URL `https://www.ecmwf.int/node/16928`.

[7] Ø. Breivik, A. Allen, C. Maisondieu, and M. Olagnon. Advances in search and rescue at sea. *Ocean Dynamics*, 63(1):83–88, Jan 2013. ISSN 1616-7228. doi: 10.1007/s10236-012-0581-1.

[8] A. Brodtkorb. *Scientific computing on heterogeneous architectures*. PhD thesis, University of Oslo, 2010.

[9] A. Brodtkorb and M. Sætra. Explicit shallow water simulations on GPUs: Guidelines and best practices. In *XIX International Conference on Water Resources, CMWR 2012, June 17–22, 2012*. University of Illinois at Urbana-Champaign, 2012.

[10] A. Brodtkorb, C. Dyken, T. Hagen, J. Hjelmervik, and O. Storaasli. State-of-the-art in heterogeneous computing. *Scientific Programming*, 18(1):1 – 33, May 2010.

[11] A. Brodtkorb, T. Hagen, K.-A. Lie, and J. Natvig. Simulation and visualization of the Saint-Venant system using GPUs. *Computing and Visualization in Science*, 13(7):1–13, 2011. ISSN 1432-9360. doi: 10.1007/s00791-010-0149-x.

[12] A. Brodtkorb, M. Sætra, and M. Altinakar. Efficient shallow water simulations on GPUs: Implementation, visualization, verification, and validation. *Computers & Fluids*, 55(0):1–12, 2012. ISSN 0045-7930. doi: 10.1016/j.compfluid.2011.10.012.

[13] G. Broström, A. Melsom, M. Sayed, and I. Kubat. Iceberg modeling at met.no: Validation of iceberg model. Technical report, Norwegian Meteorological Institute, 2009.

[14] G. Burgers, P. van Leeuwen, and G. Evensen. Analysis scheme in the ensemble Kalman filter. *Monthly Weather Review*, 126(6):1719–1724, 1998. doi: 10.1175/1520-0493(1998)126⟨1719:ASITEK⟩2.0.CO;2.

[15] M. Castro, J. López, and C. Parés. Finite volume simulation of the geostrophic adjustment in a rotating shallow-water system. *SIAM Journal on Scientific Computing*, 31(1):444—-477, 2008. ISSN 1064-8275. doi: 10.1137/070707166.

[16] M. Castro, S. Ortega, and C. Parés. Well-balanced methods for the shallow water equations in spherical coordinates. *Computers & Fluids*, 157:196–207, 2017. ISSN 0045-7930. doi: 10.1016/j.compfluid.2017.08.035.

[17] E. Chassignet, L. Smith, G. Halliwell, and R. Bleck. North Atlantic simulations with the Hybrid Coordinate Ocean Model (HYCOM): Impact of the vertical coordinate choice, reference pressure, and thermobaricity. *Journal of Physical Oceanography*, 33(12):2504–2526, 2003. doi: 10.1175/1520-0485(2003)033⟨2504:NASWTH⟩2.0.CO;2.

[18] A. Chertock, M. Dudzinski, A. Kurganov, and M. Lukácová-Medvidová. Well-balanced schemes for the shallow water equations with Coriolis forces. *Numerische Mathematik*, 2017. ISSN 0945-3245. doi: 10.1007/s00211-017-0928-0.

[19] A. Chertock, M. Dudzinski, A. Kurganov, and M. Lukácová-Medvidová. Well-balanced schemes for the shallow water equations with Coriolis forces. *Numerische Mathematik*, Dec 2017. ISSN 0945-3245. doi: 10.1007/s00211-017-0928-0.

[20] A. Chorin, M. Morzfeld, and X. Tu. *A survey of implicit particle filters for data assimilation*, pages 63–88. Springer New York, New York, NY, 2013. ISBN 978-1-4614-7789-1. doi: 10.1007/978-1-4614-7789-1_3.

[21] K. Christensen, Ø. Breivik, K.-F. Dagestad, J. Röhrs, and B. Ward. Short-term predictions of oceanic drift. *Oceanography*, 31(3):59–67, September 2018. doi: 10.5670/oceanog.2018.310.

[22] M. Courbariaux, Y. Bengio, and J.-P. David. Training deep neural networks with low precision multiplications, 2014.

[23] K.-F. Dagestad, J. Röhrs, Ø. Breivik, and B. Ådlandsvik. OpenDrift v1.0: a generic framework for trajectory modelling. *Geoscientific Model Development*, 11(4):1405–1420, 2018. doi: 10.5194/gmd-11-1405-2018.

[24] M. de la Asunción, J. Mantas, and M. Castro. Simulation of one-layer shallow water systems on multicore and CUDA architectures. *The Journal of Supercomputing*, 58(2):206–214, Nov 2011. ISSN 1573-0484. doi: 10.1007/s11227-010-0406-2.

[25] T. Dong, V. Dobrev, T. Kolev, R. Rieben, S. Tomov, and J. Dongarra. A step towards energy efficient computing: Redesigning a hydrodynamic application on CPU-GPU. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 972–981. IEEE, 2014.

[26] A. Doucet, S. Godsill, and C. Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10(3):197–208, Jul 2000. ISSN 1573-1375. doi: 10.1023/A:1008935410038.

[27] G. Evensen. Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast error statistics. *Journal of Geophysical Research: Oceans*, 99(C5):10143–10162, 1994. doi: 10.1029/94JC00572.

[28] G. Evensen. *Data Assimilation: The Ensemble Kalman Filter*. Springer Berlin Heidelberg, 2006. ISBN 9783540383017.

[29] W. Feng and T. Scogland. Top 500 supercomputer sites. `http://www.top500.org/green500/`, November 2019.

[30] A. Gill. *Atmosphere-Ocean Dynamics*. International Geophysics. Academic Press, 1982. ISBN 9780080570525.

[31] S. Gottlieb and C.-W. Shu. Total variation diminishing Runge-Kutta schemes. *Mathematics of Computation*, 67(221):73–85, jan 1998. ISSN 0025-5718. doi: 10.1090/S0025-5718-98-00913-2.

[32] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1737–1746, Lille, France, 07–09 Jul 2015. PMLR.

[33] T. Hagen, J. Hjelmervik, K.-A. Lie, J. Natvig, and M. Henriksen. Visual simulation of shallow-water waves. *Simulation Modelling Practice and Theory*, 13(8):716 – 726, 2005. ISSN 1569-190X. doi: 10.1016/j.simpat.2005.08.006. Programmable Graphics Hardware.

[34] T. Hagen, M. Henriksen, J. Hjelmervik, and K.-A. Lie. *How to solve systems of conservation laws numerically using the graphics processor as a high-performance computational engine*, pages 211–264. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-68783-2. doi: 10.1007/978-3-540-68783-2_8.

[35] T. Haiden, M. Dahoui, B. Ingleby, P. de Rosnay, C. Prates, E. Kuscu, T. Hewson, L. Isaksen, D. Richardson, H. Zuo, and L. Jones. Use of in situ surface observations at ECMWF. *ECMWF Technical Memoranda*, (834), 11 2018. doi: 10.21957/dj9lpy4wa.

[36] A. Harten. High resolution schemes for hyperbolic conservation laws. *Journal of Computational Physics*, 49(3):357 – 393, 1983. ISSN 0021-9991. doi: 10.1016/0021-9991(83)90136-5.

[37] S. Hatfield, A. Subramanian, T. Palmer, and P. Düben. Improving weather forecast skill through reduced-precision data assimilation. *Monthly Weather Review*, 146(1):49–62, 2018. doi: 10.1175/MWR-D-17-0132.1.

[38] H. Holm, A. Brodtkorb, and M. Sætra. Performance and energy efficiency of CUDA and OpenCL for GPU computing using Python. *Advances in Parallel Computing*, 36:593–604, 2020. doi: 10.3233/APC200089.

[39] Z. Horváth, R. Perdigão, J. Waser, D. Cornel, A. Konev, and G. Blöschl. Kepler shuffle for real-world flood simulations on GPUs. *The International Journal of High Performance Computing Applications*, 30(4):379–395, 2016. doi: 10.1177/1094342016630800.

[40] P. Houtekamer and H. Mitchell. Data assimilation using an ensemble Kalman filter technique. *Monthly Weather Review*, 126(3):796–811, 1998. doi: 10.1175/1520-0493(1998)126⟨0796:DAUAEK⟩2.0.CO;2.

[41] P. Houtekamer and H. Mitchell. A sequential ensemble Kalman filter for atmospheric data assimilation. *Monthly Weather Review*, 129(1):123–137, 2001. doi: 10.1175/1520-0493(2001)129⟨0123:ASEKFF⟩2.0.CO;2.

[42] S. Huang, S. Xiao, and W.-C. Feng. On the energy efficiency of graphics processing units for scientific computing. In *2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–8. IEEE, 2009.

[43] J. Hunter. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

[44] Khronos OpenCL Working Group. The OpenCL specification v. 2.2, 2018.

[45] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih. PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation. *Parallel Computing*, 38(3):157–174, 2012. ISSN 0167-8191. doi: 10.1016/j.parco.2011.09.001.

[46] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, and Jupyter development team. Jupyter notebooks – a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90, 2016. doi: 10.3233/978-1-61499-649-1-87.

[47] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012. doi: 10.1145/3065386.

[48] A. Kurganov and D. Levy. Central-upwind schemes for the Saint-Venant system. *Mathematical Modelling and Numerical Analysis*, 36:397–425, 2002.

[49] A. Kurganov and G. Petrova. A second-order well-balanced positivity preserving central-upwind scheme for the Saint-Venant system. *Communications in Mathematical Sciences*, 5(1):133–160, 03 2007.

[50] A. Kurganov, S. Noelle, and G. Petrova. Semidiscrete central-upwind schemes for hyperbolic conservation laws and Hamilton-Jacobi equations. *SIAM Journal on Scientific Computing*, 23(3):707–740, 2001. doi: 10.1137/S1064827500373413.

[51] E. Larsen and D. McAllister. Fast matrix multiplies using graphics hardware. In *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, SC '01, pages 55–55, New York, NY, USA, 2001. ACM. doi: 10.1145/582034.582089.

[52] D. Lea, J.-P. Drecourt, K. Haines, and M. Martin. Ocean altimeter assimilation with observational- and model-bias correction. *Quarterly Journal of the Royal Meteorological Society*, 134(636):1761–1774, 2008. doi: 10.1002/qj.320.

[53] R. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2004.

[54] C. Liu, Q. Xiao, and B. Wang. An ensemble-based four-dimensional variational data assimilation scheme. Part I: Technical formulation and preliminary test. *Monthly Weather Review*, 136(9):3363–3373, 2008. doi: 10.1175/2008MWR2312.1.

[55] G. Madec and the NEMO team. *NEMO ocean engine*. Note du Pôle de modélisation, Institut Pierre-Simon Laplace (IPSL), France, No 27, ISSN No 1288-1619, 2008.

[56] E. Martinsen, B. Gjevik, and L. Røed. A numerical model for long barotropic waves and storm surges along the western coast of Norway. *Journal of Physical Oceanography*, 9:1126–1138, 11 1979. doi: 10.1175/1520-0485(1979)009⟨1126:ANMFLB⟩2.0.CO;2.

[57] F. Mesinger and A. Arakawa. *Numerical Methods Used in Atmospheric Models*, volume 1 of *GARP publications series*. World Meteorological Organization, International Council of Scientific Unions, 1976.

[58] H. Meuer, E. Strohmaier, J. Dongarra, H. Simon, and M. Mauer. Top 500 supercomputer sites. `http://www.top500.org/`, November 2019.

[59] K. Mogensen and A. Balmaseda M., Weaver. The NEMOVAR ocean data assimilation system as implemented in the ecmwf ocean analysis for System 4. (668):59, 02 2012. doi: 10.21957/x5y9yrtm.

[60] A. Moore, H. Arango, G. Broquet, B. Powell, A. Weaver, and J. Zavala-Garay. The Regional Ocean Modeling System (ROMS) 4-dimensional variational data assimilation systems: Part I – system overview and formulation. *Progress in Oceanography*, 91(1):34 – 49, 2011. ISSN 0079-6611. doi: 10.1016/j.pocean.2011.05.004.

[61] J. Natvig. *High-resolution methods for conservation laws in the geosciences*. PhD thesis, University of Oslo, 2006.

[62] NVIDIA. NVIDIA CUDA C programming guide version 10.1, 2019.

[63] NVIDIA whitepaper. NVIDIA Tesla V100 GPU architecture. `https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf`, Aug. 2017. Website, retrieved 2019-12-02.

[64] J. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. Lefohn, and T. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80–113, 2007. doi: 10.1111/j.1467-8659.2007.01012.x.

[65] J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, and J. Phillips. GPU computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

[66] N. Pankratz, J. Natvig, B. Gjevik, and S. Noelle. High-order well-balanced finite-volume schemes for barotropic flows. development and numerical comparisons, 2014.

[67] P. Parna, K. Meyer, and R. Falconer. GPU driven finite difference WENO scheme for real time solution of the shallow water equations. *Computers & Fluids*, 161:107 – 120, 2018. ISSN 0045-7930. doi: 10.1016/j.compfluid.2017.11.012.

[68] D. Pham. Stochastic methods for sequential data assimilation in strongly nonlinear systems. *Monthly Weather Review*, 129(5):1194–1207, 2001. doi: 10.1175/1520-0493(2001)129⟨1194:SMFSDA⟩2.0.CO;2.

[69] N. Phillips. An example of non-linear computational instability. *The Atmosphere and the Sea in motion*, 501, 1959.

[70] Z. Qi, W. Wen, W. Meng, Y. Zhang, and L. Shi. An energy efficient OpenCL implementation of a fingerprint verification system on heterogeneous mobile device. In *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 1–8. IEEE, 2014.

[71] X. Qin, R. LeVeque, and M. Motley. Accelerating an adaptive mesh refinement code for depth-averaged flows using GPUs. *Journal of Advances in Modeling Earth Systems*, 11(8):2606–2628, 2019. doi: 10.1029/2019MS001635.

[72] S. Reich and C. Cotter. *Probabilistic Forecasting and Bayesian Data Assimilation*. Cambridge University Press, 2015.

[73] H. Roarty, T. Cook, L. Hazard, D. George, J. Harlan, S. Cosoli, L. Wyatt, E. Alvarez Fanjul, E. Terrill, M. Otero, J. Largier, S. Glenn, N. Ebuchi, B. Whitehouse, K. Bartlett, J. Mader, A. Rubio, L. Corgnati, C. Mantovani, A. Griffa, E. Reyes, P. Lorente, X. Flores-Vidal, K. J. S.-M., P. Rogowski, S. Prukpitikul, S.-H. Lee, J.-W. Lai, C.-A. Guerin, J. Sanchez, B. Hansen, and S. Grilli. The global high frequency radar network. *Frontiers in Marine Science*, 6:164, 2019. ISSN 2296-7745. doi: 10.3389/fmars.2019.00164.

[74] A. Robert, F. Shuman, and J. Gerrity. On partial difference equations in mathematical physics. *Monthly Weather Review*, 98(1):1–6, 1970. doi: 10.1175/1520-0493(1970)098⟨0001:OPDEIM⟩2.3.CO;2.

[75] S. Robert and H. Künsch. Localizing the ensemble Kalman particle filter. *Tellus A: Dynamic Meteorology and Oceanography*, 69(1):1282016, 2017. doi: 10.1080/16000870.2017.1282016.

[76] L. Røed. Documentation of simple ocean models for use in ensemble predictions. Part I: Theory. Technical report, Norwegian Meteorological Institute, 2012.

[77] L. Røed. *Atmospheres and Oceans on Computers*. Springer International Publishing, 2019. doi: 10.1007/978-3-319-93864-6.

[78] J. Röhrs, K.-F. Dagestad, H. Asbjørnsen, T. Nordam, J. Skancke, C. E. Jones, and C. Brekke. The effect of vertical mixing on the horizontal drift of oil spills. *Ocean Science*, 14(6):1581–1601, 2018. doi: 10.5194/os-14-1581-2018.

[79] J. Röhrs, A. Sperrevik, and K. Christensen. NorShelf: A reanalysis and data-assimilative forecast model for the Norwegian Shelf Sea. Technical Report 4, Norwegian Meteorological Institute, 2018.

[80] F. Roquet, C. Wunsch, G. Forget, P. Heimbach, C. Guinet, G. Reverdin, J.-B. Charrassin, F. Bailleul, D. Costa, L. Huckstadt, K. Goetz, K. Kovacs, C. Lydersen, M. Biuw, O. Nøst, H. Bornemann, J. Ploetz, M. Bester, T. McIntyre, M. Muelbert, M. Hindell, C. McMahon, G. Williams, R. Harcourt, I. Field, L. Chafik, K. Nicholls, L. Boehme, and M. Fedak. Estimates of the Southern Ocean general circulation improved by animal-borne instruments. *Geophysical Research Letters*, 40(23):6176–6180, 2013. doi: 10.1002/2013GL058304.

[81] J. Röhrs and K. Christensen. Drift in the uppermost part of the ocean. *Geophysical Research Letters*, 42(23):10,349–10,356, 2015. doi: 10.1002/2015GL066733.

[82] M. Sætra. *Shallow water simulations on graphics hardware*. PhD thesis, University of Oslo, 2014.

[83] M. Sætra, A. Brodtkorb, and K.-A. Lie. Efficient GPU-implementation of adaptive mesh refinement for the shallow-water equations. *Journal of Scientific Computing*, 63(1):23–48, Apr. 2015. ISSN 0885-7474. doi: 10.1007/s10915-014-9883-4.

[84] J. Sanders and E. Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 2010.

[85] C. Schmid, R. Molinari, R. Sabina, Y.-H. Daneshzadeh, X. Xia, E. Forteza, and H. Yang. The real-time data management system for Argo profiling float observations. *Journal of Atmospheric and Oceanic Technology*, 24(9):1608–1628, 2007. doi: 10.1175/JTECH2070.1.

[86] A. Shchepetkin and J. McWilliams. The regional oceanic modeling system (ROMS): A split-explicit, free-surface, topography-following-coordinate oceanic model. *Ocean Modelling*, 9(4):347–404, 2005. ISSN 1463-5003. doi: 10.1016/j.ocemod.2004.08.002.

[87] A. Sielecki. An energy-conserving difference scheme for the storm surge equations. *Monthly Weather Review*, 96(3):150–156, 1968. doi: 10.1175/1520-0493(1968)096⟨0150:AECDSF⟩2.0.CO;2.

[88] J. Skauvold, J. Eidsvik, P. van Leeuwen, and J. Amezcua. A revised implicit equal-weights particle filter. *Quarterly Journal of the Royal Meteorological Society*, 145(721):1490–1502, 2019. doi: 10.1002/qj.3506.

[89] C. Snyder, T. Bengtsson, P. Bickel, and J. Anderson. Obstacles to high-dimensional particle filtering. *Monthly Weather Review*, 136(12):4629–4640, 2008. doi: 10.1175/2008MWR2529.1.

[90] C. Snyder, T. Bengtsson, and M. Morzfeld. Performance bounds for particle filters using the optimal proposal. *Monthly Weather Review*, 143(11):4750–4761, 2015. doi: 10.1175/MWR-D-15-0144.1.

[91] Statistisk sentralbyrå. Energy consumption in households, 2012. `https://www.ssb.no/en/energi-og-industri/statistikker/husenergi/hvert-3-aar/2014-07-14`, 2014. Website, retrieved 2019-12-09.

[92] P. Sweby. High resolution schemes using flux limiters for hyperbolic conservation laws. *SIAM Journal on Numerical Analysis*, 21(5):995–1011, 1984. doi: 10.1137/0721062.

[93] E. Toro. *Shock-Capturing Methods for Free-Surface Shallow Flows*. John Wiley & Sons, Ltd., 2001.

[94] P. van Leeuwen. Particle filtering in geophysical systems. *Monthly Weather Review*, 137(12):4089–4114, 2009. doi: 10.1175/2009MWR2835.1.

[95] P. van Leeuwen, L. Nerger, R. Potthast, S. Reich, and H. Kunsch. A review of particle filters for geoscience applications. *Quarterly Journal of the Royal Meteorological Society*, 2019. doi: 10.1002/qj.3551.

[96] F. Váňa, P. Düben, S. Lang, T. Palmer, M. Leutbecher, D. Salmond, and G. Carver. Single precision in weather forecasting models: An evaluation with the IFS. *Monthly Weather Review*, 145(2):495–502, 2017.

[97] S. Vetra-Carvalho, P. J. van Leeuwen, L. Nerger, A. Barth, M. U. Altaf, P. Brasseur, P. Kirchgessner, and J.-M. Beckers. State-of-the-art stochastic data assimilation methods for high-dimensional non-Gaussian problems. *Tellus A: Dynamic Meteorology and Oceanography*, 70(1):1–43, 2018. doi: 10.1080/16000870.2018.1445364.

[98] P. Watson. Applying machine learning to improve simulations of a chaotic dynamical system using empirical error correction. *Journal of Advances in Modeling Earth Systems*, 11(5):1402–1417, 2019. doi: 10.1029/2018MS001597.

[99] G. Wilson, D. Aruliah, C. Brown, N. Chue Hong, M. Davis, R. Guy, S. Haddock, K. Huff, I. Mitchell, M. Plumbley, B. Waugh, E. White, and P. Wilson. Best practices for scientific computing. *PLOS Biology*, 12(1):1–7, 2014. doi: 10.1371/journal.pbio.1001745.

[100] J. Xie, L. Bertino, F. Counillon, K. Lisæter, and P. Sakov. Quality assessment of the TOPAZ4 reanalysis in the Arctic over the period 1991–2013. *Ocean Science*, 13(1):123–144, 2017. doi: 10.5194/os-13-123-2017.

[101] M. Zhu, P. J. van Leeuwen, and J. Amezcua. Implicit equal-weights particle filter. *Quarterly Journal of the Royal Meteorological Society*, 142(698):1904–1919, 2016. doi: 10.1002/qj.2784.

[102] Z.-N. Zhu, X.-H. Zhu, X. Guo, X. Fan, and C. Zhang. Assimilation of coastal acoustic tomography data using an unstructured triangular grid ocean model for water with complex coastlines and islands. *Journal of Geophysical Research: Oceans*, 122(9):7013–7030, 2017. doi: 10.1002/2017JC012715.

Part II

Scientific Papers

Paper I

Evaluation of Selected Finite-Difference and Finite-Volume Approaches to Rotational Shallow-Water Flow

Håvard Heitlo Holm, André Rigland Brodtkorb, Göran Broström, Kai H. Christensen, Martin Lilleeng Sætra

Communications in Computational Physics, volume 27, number 4, pages 1234–1274, 2020
DOI: 10.4208/cicp.OA-2019-0033

Evaluation of Selected Finite-Difference and Finite-Volume Approaches to Rotational Shallow-Water Flow

Håvard Heitlo Holm*[1,3], André R. Brodtkorb[1,4], Göran Broström[2,5], Kai H. Christensen[2,6], and Martin L. Sætra[2,4]

[1] SINTEF Digital, Mathematics and Cybernetics, P.O. Box 124 Blindern, NO-0314 Oslo, Norway.
[2] Norwegian Meteorological Institute, P.O. Box 43 Blindern, NO-0313 Oslo, Norway.
[3] Norwegian University of Science and Technology, Department of Mathematical Sciences, NO-7491 Trondheim, Norway.
[4] Oslo Metropolitan University, Department of Computer Science, P.O. Box 4 St. Olavs plass, NO-0130 Oslo, Norway.
[5] University of Gothenburg, Department of Marine Sciences, P.O. Box 461, SE-405 30 Göteborg, Sweden.
[6] University of Oslo, Department of Geosciences, P.O. Box 1047 Blindern, NO-0316 Oslo, Norway

Abstract

The shallow-water equations in a rotating frame of reference are important for capturing geophysical flows in the ocean. In this paper, we examine and compare two traditional finite-difference schemes and two modern finite-volume schemes for simulating these equations. We evaluate how well they capture the relevant physics for problems such as storm surge and drift trajectory modelling, and the schemes are put through a set of six test cases. The results are presented in a systematic manner through several tables, and we compare the qualitative and quantitative performance from a cost-benefit perspective. Of the four schemes, one of the traditional finite-difference schemes performs best in cases dominated by geostrophic balance, and one of the modern finite-volume schemes is superior for capturing gravity-driven motion. The traditional finite-difference schemes are significantly faster computationally than the modern finite-volume schemes.

1 Introduction

In this paper, we examine four different numerical schemes for the shallow-water equations in a rotating frame. These equations are important for a range of application areas, including simulation of the ocean and atmosphere. We focus on oceanographic simulations, in which the equations can capture the short-term ocean dynamics that are important for e.g., storm surge predictions. Our aim is to evaluate the suitability of the numerical schemes for use in an ensemble prediction system with data assimilation. One example is the propagation of long waves in an ocean basin sufficiently large so that the motion is constrained by geostrophy, and where we need to consider the effects of topography and nonlinearity, e.g. in the Barents Sea, which is fairly shallow but have large tidal range. We emphasize that we do not seek realistic solutions of the ocean dynamics for specific regions here, but rather aim to compare the various schemes using a range of parameters relevant for such dynamics.

In the early days of computational oceanography, finite-difference schemes were popular to simulate the rotational shallow-water equations. With increasing computational power, more complex physics, grids, and new discretization methods have appeared. Today's state-of-the-art ocean circulation models are sophisticated 3D simulations that capture a lot of the physical driving forces of the ocean currents, yet these models are computationally demanding and therefore allow only a limited number of ensemble members to be run in reasonable time.

We revisit two finite-difference schemes from early computational oceanography and compare these against two modern finite-volume schemes. One of our motivations for comparing finite-volume and finite-difference discretizations

*Corresponding author: havard.heitlo.holm@sintef.no is that there has been recent developments in the finite-volume community for the rotating shallow-water equations, and we want to evaluate these from a cost-benefit perspective against well-known models.

The dominant force balance in the equations implies a nonzero current as a steady state, as the pressure gradient needs to be balanced by the Coriolis forces (so-called geostrophic balance). This is very different from problems in which the Earth's rotation can be ignored, where a typical steady state would imply zero velocities, often referred to as "lake-at-rest". This difference has important implications for the discretization of numerical schemes, and has been one of the driving factors in the development of modern high-resolution finite-volume methods which are well-balanced according to such steady-state solutions.

The four selected schemes in this work are all based on Cartesian grids, and are selected both because they capture the important geostrophic balance required for short-term predictions, and because they are very well suited for implementation on the GPU. Our long-term goal is to run large ensembles of such models, initialized and downscaled from operational 3D circulation models, on the GPU. These large ensembles can then be used on-demand to provide uncertainty estimates in predictions of storm surge or in drift trajectory modelling. The four numerical schemes we examine are:

FBL The Forward-Backward-Linear (FBL) finite-difference scheme on an Arakawa C grid [1] is based on a linearization of the shallow-water equations, and can therefore only represent the linear physics of ocean circulation. However, due to its simplicity it is very efficient and is therefore included to be evaluated from a cost-benefit perspective.

CTCS The Centered-in-Time Centered-in-Space (CTCS) scheme is a classic finite-difference leapfrog scheme on an Arakawa C grid. It is arguably one of the simplest numerical schemes used for geophysical flows, and is well-known and suited for geostrophically balanced flows. It is hence adequate as a reference to compare the finite-volume schemes against.

KP The high-resolution finite-volume Kurganov-Petrova scheme [2] has traditionally been used for modelling fast waves in non-rotating reference frames (e.g., dam break problems), and efficient implementations on heterogeneous platforms have been demonstrated [3]. In this work, we have added a naïve discretization of the Coriolis force, and the scheme illustrates the benefits and drawbacks of such an approach.

CDKLM The high-resolution finite-volume scheme introduced by Chertock et al. [4] is similar to the KP scheme, but it is specifically tailored to capture steady states which are in geostrophic balance. It represents a modern finite-volume scheme tailored to oceanographic applications.

We have evaluated these numerical schemes using a set of six test cases that contain important components of the transient barotropic ("fast", see Section 2) dynamics. The test cases are relevant for shelf seas or basin scale applications, and include both fast moving (Kelvin) and slow moving (Rossby) waves. The latter is particularly interesting since we obtain essentially the same wave phenomena from latitudinal variations in the Coriolis parameter compared with equivalent variations in topography. This provides a robust test of the higher-order schemes.

The rest of this paper is structured as follows. The remainder of this section mentions some relevant related work. Section 2 introduces the rotational shallow-water equations in an oceanography setting and describes the mathematical model from both the oceanographic and the finite volume perspective. Section 3 outlines the selected numerical schemes, followed by a detailed description of the test cases and results in Section 4. Finally, the paper is summarized in Section 5. As we anticipate some readers may not be familiar with all concepts of geophysical fluid dynamics, we provide some underlying theory in A.

The software used herein has been designed to be suitable for a high-performance ensemble prediction system with non-linear data assimilation. The test cases we use are comprehensively described and tailored to be reproducible for other researchers using the current simulation framework or other codes, and the numerical schemes are implemented on the GPU for efficiency. Both the full source code and relevant test case setups are available as supplementary material.

Related work

There has been a substantial effort in designing numerical schemes for the shallow-water equations using various approximation methods (e.g., finite volume [5, 6], finite difference [7, 8], lattice Boltzmann [9], and discontinuous Galerkin [10, 11]). The finite-volume community has recently shown an interest in rotational shallow-water flows (i.e., flows under the influence of the Coriolis force). This has led to the development of schemes that is well-balanced with respect to the geostrophic balance [4], and not only with respect to the lake-at-rest solution [12]. Furthermore, some of these higher-order numerical schemes are particularly suitable for implementation on massively data parallel architectures like the GPU [13, 14, 15, 3, 16]. Some mature software packages, such as Clawpack [17], also have GPU implementations of some finite-volume schemes [18], and several commercial packages, such as TUFLOW [19] and MIKE [20], have GPU support.

An implementation of any numerical scheme should be verified against appropriate analytical solutions, or validated against specially designed test cases or real-world data. For the non-rotating shallow-water equations, there are several well-established reference solutions for specific physical phenomena, see e.g., [21, 22, 23, 24, 25, 26, 27, 28]. In the case of rotating shallow-water flow, a notable test set is described by Williamson et al. [29], and includes both analytic and high-resolution reference solutions to seven test cases defined on the sphere. The test set has also been suggested extended by Galewsky et al. [30]. Comblen et al. [31] used a set of eight test cases for evaluating five different pairs of finite-element methods for solving the rotational shallow-water equations on a flat two-dimensional domain. A similar approach with six test cases was used by Tumolo et al. [32] to evaluate their discontinuous Galerkin method for solving the same problem.

2 The Rotating Shallow-Water Equations

The shallow-water equations describe flows in "shallow" water, meaning that the horizontal scales of the problem are much larger than the water depth, and are here considered on a rotating domain. The equations can describe important physical processes found in the ocean, such as tide propagation, storm surge, and wave phenomena such as inertial oscillations, Kelvin waves, and Rossby waves induced by changes in bottom topography or in the Coriolis force. It is commonly assumed that the pressure distribution is hydrostatic and that the vertical acceleration can be neglected. The pressure at any specific point is then simply a function of the water density and the height of the water column above it. For some physical processes, such as large-amplitude or higher-mode internal waves, the hydrostatic approximation cannot be used, but in general the approximations involved do not lead to significant errors for barotropic flows. Strictly speaking, barotropic motion is characterized by coincidental surfaces of constant pressure and constant density, but can be thought of as motion with negligible vertical shear. The opposite case gives rise to baroclinic motion, and examples here include internal waves and instabilities that can develop into ocean eddies. In general, barotropic signals propagate much faster than baroclinic signals, which is a challenge in numerical ocean circulation models, because the temporal resolution required for numerical stability can differ by one or two orders of magnitude. This problem is typically overcome by using so-called mode splitting, with different time-step sizes for the integration of the barotropic and baroclinic components, respectively (e.g., [33]).

The shallow-water equations are often expressed differently by those working with hyperbolic conservation laws and those working with ocean modelling, even though the starting point in both cases is the Navier-Stokes equations. Both communities use equations in flux form that are essentially hyperbolic, but in the oceanographic community it is customary to keep all forcing terms separate from the conservation of momentum. On the other hand, the community working with hyperbolic conservation laws typically include parts of the forces due to pressure gradients in the flux term, and this results in a different set of forcing terms in the equations.

In this section, both formulations of the shallow-water equations are presented, and we show how they are equivalent. For simplicity, viscous terms due to surface and bottom stresses, and variations in the atmospheric forcing are ignored. Both formulations are essentially based on vertical integration of the governing equations, with kinematic boundary conditions prescribing no flow normal to fixed boundaries and a moving free surface.

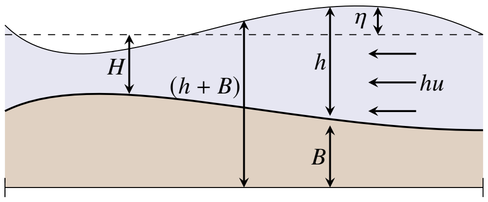


Figure 1: Relationship between variables h, H, η, hu and B, appearing in the two formulations of the shallow-water equations, here shown in one dimension.

2.1 Classical Formulation as Hyperbolic Conservation Laws

In the context of hyperbolic conservation laws (e.g., [6]), the shallow-water equations are derived by considering conservation of mass and momentum. The starting point is the Navier-Stokes equations, which are depth integrated under the assumptions mentioned above. Among those working with finite-volume methods for hyperbolic conservation laws, the shallow-water equations are often written as

$$\begin{bmatrix} h \\ hu \\ hv \end{bmatrix}_t + \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{bmatrix}_x + \begin{bmatrix} hv \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix}_y = \begin{bmatrix} 0 \\ fhv \\ -fhu \end{bmatrix} + \begin{bmatrix} 0 \\ -ghB_x \\ -ghB_y \end{bmatrix}. \tag{1}$$

Here, h is water depth, and u and v denote water velocity in x- and y-direction, respectively. The conserved variables hu and hv represent the volume transport, also referred to as discharge or momentum. The source terms on the right-hand-side represent the Coriolis force caused by the rotation of the Earth and acceleration due to gravity over a varying bathymetry. The bathymetry is described by $B(x, y)$ as elevation above a reference level.

In the present applications, water depth h can often be in the range of 1000 meters, while the change in surface elevation across time-steps is typically only a few centimeters, resulting in five orders of magnitude difference. For better numerical representation, we can rewrite the problem in terms of the deviation η from equilibrium water depth. Denoting equilibrium water depth as H, so that $h = \eta + H$, (1) becomes

$$\begin{bmatrix} \eta \\ hu \\ hv \end{bmatrix}_t + \begin{bmatrix} hu \\ \frac{(hu)^2}{H+\eta} + \frac{1}{2}g(H+\eta)^2 \\ \frac{(hu)(hv)}{H+\eta} \end{bmatrix}_x + \begin{bmatrix} hv \\ \frac{(hu)(hv)}{H+\eta} \\ \frac{(hv)^2}{H+\eta} + \frac{1}{2}g(H+\eta)^2 \end{bmatrix}_y = \begin{bmatrix} 0 \\ fhv \\ -fhu \end{bmatrix} + \begin{bmatrix} 0 \\ g(H+\eta)H_x \\ g(H+\eta)H_y \end{bmatrix}. \tag{2}$$

Here, we have used the fact that H is independent of time and that the slope of H and B have opposite signs. Relationships between all the introduced variables are shown in Figure 1 for the one-dimensional case.

By denoting the vector of conserved varibles as $\boldsymbol{q} = [\eta, hu, hv]^T$, we can write (2) in vector form as

$$\boldsymbol{q}_t + F(\boldsymbol{q})_x + G(\boldsymbol{q})_y = S_f(\boldsymbol{q}) + S_H(\boldsymbol{q}, \nabla H), \tag{3}$$

in which F and G represent fluxes along the abscissa and ordinate, respectively, and S_f and S_H are the Coriolis and bed slope source terms, respectively.

2.2 Classical Formulation in Physical Oceanography

Within the physical oceanography community, the shallow-water equations are typically derived by using the continuity equation along with the Navier-Stokes equations [7]. After vertical integration of the governing equations and application of the kinematic boundary conditions, we end up with the following form for the shallow-water equations:

$$\eta_t + \nabla_H \cdot \boldsymbol{U} = 0, \tag{4}$$

$$\boldsymbol{U}_t + \nabla_H \cdot \left(\frac{\boldsymbol{U}\boldsymbol{U}}{h}\right) = -gh\nabla_H\eta - f\boldsymbol{k} \times \boldsymbol{U} + A\nabla_H^2\boldsymbol{U}. \tag{5}$$

The horizontal divergence is denoted $\nabla_H = [\boldsymbol{i}\frac{\partial}{\partial x}, \boldsymbol{j}\frac{\partial}{\partial y}, 0]^T$. Here, $\boldsymbol{i}$ and $\boldsymbol{j}$ are the horizontal unit vectors, whereas $\boldsymbol{k}$ is the vertical unit vector. Further, the vector $\boldsymbol{U}$ is the depth integrated volume transport, $\boldsymbol{U} = [hu, hv, 0]^T$. The final term represents an explicit parametrization of a diffusive process that is sometimes used to avoid nonlinear numerical instabilities when the equations are solved with e.g., the classical leapfrog finite-difference method considered herein. The constant A is referred to as the eddy viscocity parameter. In (4) and (5), vertical shear stresses caused by bottom friction and wind drag on the surface are ignored.

By examining the two different formulations of the shallow-water equations, we observe that (4) is equal to the first row of (2), since

$$\eta_t + \nabla_H \cdot \boldsymbol{U} = \eta_t + (hu)_x + (hv)_y = 0.$$

A closer look at the second nonlinear term of (5) reveals that

$$\nabla_H \cdot \left(\frac{\boldsymbol{U}\boldsymbol{U}}{h}\right) = \nabla_H \cdot \frac{1}{h}\begin{bmatrix} (hu)^2 & (hu)(hv) & 0 \\ (hu)(hv) & (hv)^2 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} (hu^2)_x + (huv)_y \\ (huv)_x + (hv^2)_y \\ 0 \end{bmatrix},$$

in which we can recognize the non-gravity driven flux terms from the second and third row of (1). Next, we consider the gravity induced flux terms and the bathymetry source term in x-direction from (1), moving the source term to the left-hand side. By using $h = \eta + H$ and the fact that $\nabla_H H = -\nabla_H B$, these two terms can be written as

$$\left(\frac{1}{2}gh\right)_x + ghB_x = gh(\eta_x + H_x) + ghB_x = gh\eta_x,$$

which we recognise on the right-hand side of (5). The same manipulation applies in y-direction. If we ignore the eddy viscosity term, and since

$$f\boldsymbol{k} \times \boldsymbol{U} = [-f(hu), f(hv), 0]^T,$$

we see that the first and second row of (5) correspond to the second and third row of (1). The third row of (5) represents the vertical momentum, which is zero on both sides of the equation.

The eddy viscosity $A\nabla_H^2\boldsymbol{U}$ in (5) is not represented in (1). This is because it is related to stability issues in the leapfrog finite-difference scheme [34], an issue not present with the finite-volume schemes for which the formulation (1) is used.

3 Numerical Schemes

In the following, we give an overview of four different numerical schemes for solving the rotational shallow-water equations. The first two schemes are finite-difference methods, and the latter two are high-resolution finite-volume methods, in which the term high-resolution refers to the schemes' ability to accurately capture discontinuities. At the end of this section we describe three boundary conditions, and discuss the main differences between the selected schemes.

3.1 Forward-Backward Linear Scheme

The first scheme is the Forward-Backward Linear (FBL) finite-difference scheme, first presented by Sielecki [35]. It considers η, $U = hu$, and $V = hv$, in which U and V are used for the volume transport to provide a more compact

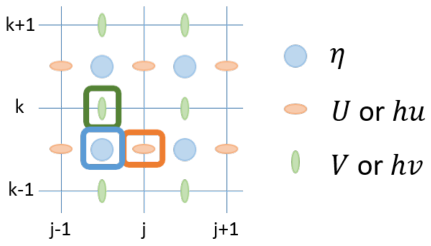


Figure 2: The discretized conserved variables on a staggered lattice C grid.

notation. The scheme is based on the linearized equations

$$\begin{aligned} \eta_t &= -U_x - V_y, \\ U_t - fV &= -gH\eta_x, \\ V_t + fU &= -gH\eta_y, \end{aligned} \tag{6}$$

arising from scalar linearization of (4) and (5). The name of the scheme reflects the first-order discretization used in time, while second-order central differences are applied in space [36].

To write out a finite-difference method for (6), consider a regular Cartesian discretization on a rectangular domain. Let Δx and Δy be the distance between each point in the grid, and define $x_j = j\Delta x$ for $j = 0, \ldots, N_x$, and $y_k = k\Delta y$ for $k = 0, \ldots, N_y$. The variables η, U, and V are defined according to a lattice C grid, as defined by Mesinger and Arakawa [1] and shown in Figure 2. The discretized variables then become

$$\begin{aligned} \eta^n_{j,k} &= \eta(x_j - \frac{1}{2}\Delta x, y_k - \frac{1}{2}\Delta y, t_n), \\ U^n_{j,k} &= U(x_j, y_k - \frac{1}{2}\Delta y, t_n), \\ V^n_{j,k} &= V(x_j - \frac{1}{2}\Delta x, y_k, t_n), \end{aligned} \tag{7}$$

in which t_n is time-step n.

The finite-difference method uses an asymmetric update in time, in which each of the variables are updated based on the most recent available state of the others. The FBL scheme is then given as

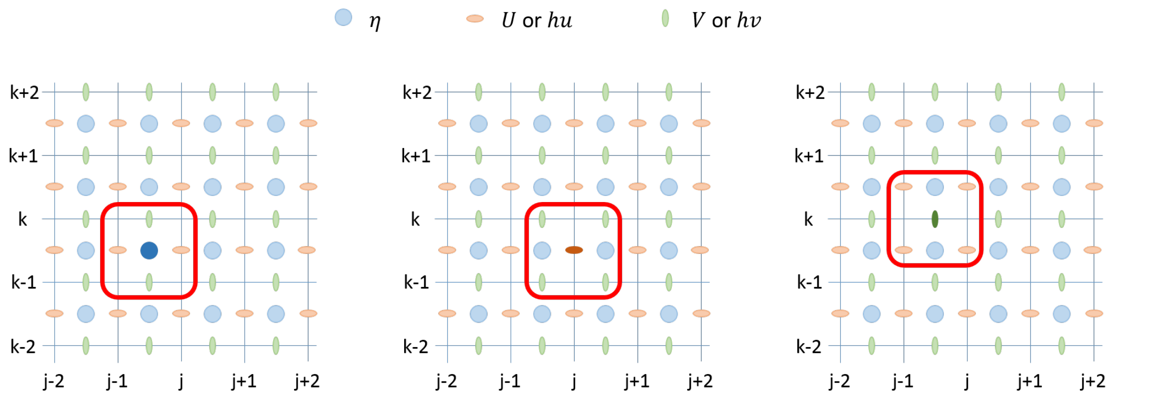
$$\begin{aligned} U^{n+1}_{j,k} &= U^n_{j,k} + \Delta t \overline{fV}^n_{j,k} + \frac{\Delta t}{2\Delta x} g \left(H_{j+1,k} + H_{j,k}\right)\left(\eta^n_{j+1,k} - \eta^n_{j,k}\right), \\ V^{n+1}_{j,k} &= V^n_{j,k} - \Delta t \overline{fU}^{n+1}_{j,k} + \frac{\Delta t}{2\Delta y} g \left(H_{j,k+1} + H_{j,k}\right)\left(\eta^n_{j,k+1} - \eta^n_{j,k}\right), \\ \eta^{n+1}_{j,k} &= \eta^n_{j,k} - \frac{\Delta t}{\Delta x}\left(U^{n+1}_{j,k} - U^{n+1}_{j-1,k}\right) - \frac{\Delta t}{\Delta y}\left(V^{n+1}_{j,k} - V^{n+1}_{j,k-1}\right), \end{aligned} \tag{8}$$

in which

$$\begin{aligned} \overline{fU}^n_{j,k} &= \frac{1}{4}\left(f_{k-\frac{1}{2}} U^n_{j,k} + f_{k-\frac{1}{2}} U^n_{j-1,k} + f_{k+\frac{1}{2}} U^n_{j-1,k+1} + f_{k+\frac{1}{2}} U^n_{j,k+1}\right), \\ \overline{fV}^n_{j,k} &= \frac{1}{4}\left(f_k V^n_{j,k} + f_k V^n_{j+1,k} + f_{k+1} V^n_{j+1,k-1} + f_{k+1} V^n_{j,k-1}\right). \end{aligned} \tag{9}$$

Since the Coriolis force varies with the latitude only, the discrete values of the Coriolis force is denoted as $f_k = f(y_k)$. The grid values required for each of these stencils are shown in Figure 3.

The scheme requires that initial conditions for η, U, and V are given at $t = t_0$, as well as boundary conditions for U at $x = 0$ and $x = N_x\Delta x$, and for V at $y = 0$ and $y = N_y\Delta y$ for $t \geq t_0$. In the ocean the fastest signals are due to



(a) Stencil for $\eta_{j,k}$. (b) Stencil for $U_{j,k}$. (c) Stencil for $V_{j,k}$.

Figure 3: Finite-difference stencils for the FBL scheme. Blue circles represent η, while orange horizontal ellipses and green vertical ellipses represent U and V, respectively. Note that the water depth H is collocated with η.

barotropic shallow-water waves, propagating with speed $|u| + \sqrt{gH_{max}}$, in which H_{max} is the maximum equilibrium water depth. Since the fluid velocities $|u|$ are usually at least one order of magnitude smaller than the gravitational term, the relevant Courant-Friedrich-Levy (CFL) condition in for FBL becomes

$$\Delta t \leq \frac{\min(\Delta x, \Delta y)}{\sqrt{2gH_{max}}}. \tag{10}$$

3.2 Centered-in-Time, Centered-in-Space Scheme

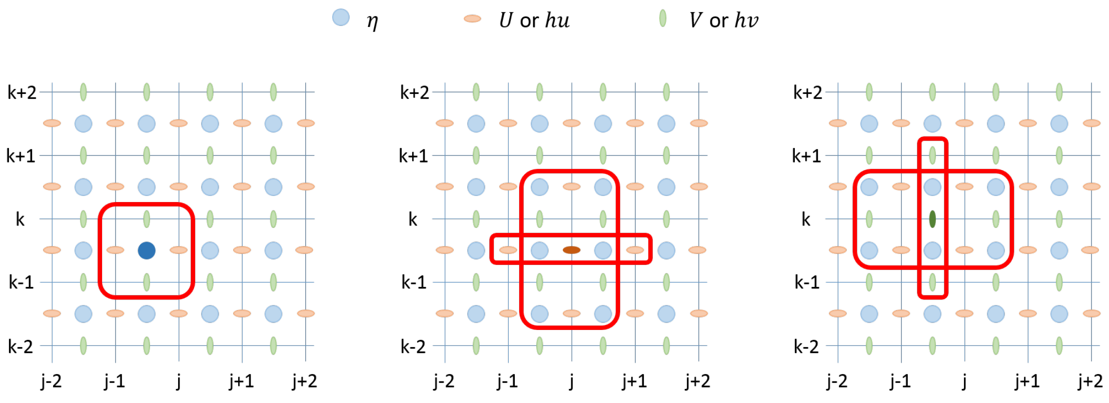
The centered-in-time, centered-in-space (CTCS) finite-difference scheme discretizes the full nonlinear equations given in (4) and (5), using the same staggered grid as FBL. It is a leapfrog scheme, and can be summarized by the stencils

$$\begin{aligned}
\eta^{n+1}_{j,k} &= \eta^{n-1}_{j,k} - \frac{2\Delta t}{\Delta x}\left(U^n_{j,k} - U^n_{j-1,k}\right) - \frac{2\Delta t}{\Delta y}\left(V^n_{j,k} - V^n_{j,k-1}\right), \\
U^{n+1}_{j,k} &= \frac{1}{B^x_{j,k}}\left[U^{n-1}_{j,k} + 2\Delta t\left(\overline{fV}^n_{j,k} + \frac{1}{\Delta x}N^x_{j,k} + \frac{1}{\Delta x}P^x_{j,k} + AE^x_{j,k}\right)\right], \\
V^{n+1}_{j,k} &= \frac{1}{B^y_{j,k}}\left[V^{n-1}_{j,k} + 2\Delta t\left(-\overline{fU}^n_{j,k} + \frac{1}{\Delta y}N^y_{j,k} + \frac{1}{\Delta y}P^y_{j,k} + AE^y_{j,k}\right)\right].
\end{aligned} \tag{11}$$

The factors $\overline{fU}$ and $\overline{fV}$ are the same as for the FBL scheme, given in (9), while the N and P terms handle the momentum fluxes and gravity pressure terms, respectively. The terms E and B are all used to handle the eddy viscosity term, and the interested reader can find details of the full scheme in Røed [36]. Unlike the FBL scheme, CTCS is symmetric in time and space, as all three variables are updated independently for each time-step, allowing all three stencils to be computed in parallel within each time-step.

The complete set of grid values required for each of these stencils is shown in Figure 4. In addition to the boundary conditions required for the FBL scheme, it is also necessary to define values for η and U at $y = -\frac{1}{2}\Delta y$ and $y = (N_y + \frac{1}{2})\Delta y$, and correspondingly for η and V at $x = -\frac{1}{2}\Delta x$ and $x = (N_x + \frac{1}{2})\Delta x$. Since the stencil also includes terms from $t = t_{n+1}$, initial conditions for both t_0 and t_{-1} are required.

The stability of the CTCS scheme is restricted under half the CFL criterion (10) as the FBL scheme [8]. Additionally, the CTCS scheme is known to suffer under nonlinear instability [37, 38, 8]. The nonlinearity of the equations allows energy to be redistributed between waves of different wave lengths, causing energy to be transferred from long waves



(a) Stencil for $\eta_{j,k}$. (b) Stencil for $U_{j,k}$. (c) Stencil for $V_{j,k}$.

Figure 4: Finite-difference stencils for the CTCS scheme. Blue circles represent η, while orange horizontal ellipses and green vertical ellipses represent U and V, respectively.

to shorter ones. As the solution on the discrete grid is unable to represent waves with wave lengths shorter than $2\Delta x$, waves with wave lengths between $2\Delta x$ and $4\Delta x$ tend to increase in amplitude, and eventually lead to unstable solutions. The eddy viscosity term, controlled empirically through the A parameter, introduces artificial diffusion into the scheme, damping the shorter waves and avoiding the solution to be dominated by strong short waves.

3.3 Kurganov-Petrova Scheme

The Kurganov-Petrova 2007 (KP) scheme [2] is a high-resolution finite-volume scheme [6], and is based on the full nonlinear equations (2). The flux terms are computed by a well-balanced, positivity-preserving, central-upwind method. This means that the flux balances the bathymetry source terms so that lake-at-rest solutions are preserved, even in the presence of discontinuous bathymetry. The scheme ensures that the water depth always is non-negative, which is important at wet-dry interfaces. The scheme has built-in numerical diffusion, and hence does not have the same problem with nonlinear instabilities due to build-up of energy at the smallest spatial scales, as is the case with CTCS.

KP can be written as

$$\begin{aligned}\frac{d\boldsymbol{Q}_{j,k}}{dt} =& S_f(\boldsymbol{Q}_{j,k}) + S_H(\boldsymbol{Q}_{j,k}, \nabla H) - \left[F(\boldsymbol{Q}_{j+1/2,k}) - F(\boldsymbol{Q}_{j-1/2,k})\right] \\ &- \left[G(\boldsymbol{Q}_{j,k+1/2}) - G(\boldsymbol{Q}_{j,k-1/2})\right], \\ :=& R(\boldsymbol{Q})_{j,k}.\end{aligned} \tag{12}$$

Here $\boldsymbol{Q}_{j,k}$ is the vector of conserved variables averaged over the grid cell centered at $((j+\frac{1}{2})\Delta x, (k+\frac{1}{2})\Delta y)$, for $j = 0, \ldots, N_x - 1$ and $k = 0, \ldots, N_y - 1$. Further, S_f and S_H are discretized Coriolis and bed slope source terms, respectively, and F and G represent numerical flux functions.

The scheme consists of the following steps: From the averaged cell values, a piecewise bilinear polynomial of $\boldsymbol{Q}$ is *reconstructed* by using the generalized minmod limiter. The slope allows us to evaluate $\boldsymbol{Q}$ at each side of every face, and the central-upwind numerical flux function is used to compute F and G. The solution is then *evolved* in time, by using a second-order, strong stability preserving Runge–Kutta method [39]:

$$\begin{aligned}\boldsymbol{Q}^*_{j,k} &= \boldsymbol{Q}^n_{j,k} + \Delta t R(\boldsymbol{Q}^n)_{j,k}, \\ \boldsymbol{Q}^{n+1}_{j,k} &= \tfrac{1}{2}\boldsymbol{Q}^n_{j,k} + \tfrac{1}{2}\left[\boldsymbol{Q}^*_{j,k} + \Delta t R(\boldsymbol{Q}^*)_{j,k}\right].\end{aligned} \tag{13}$$

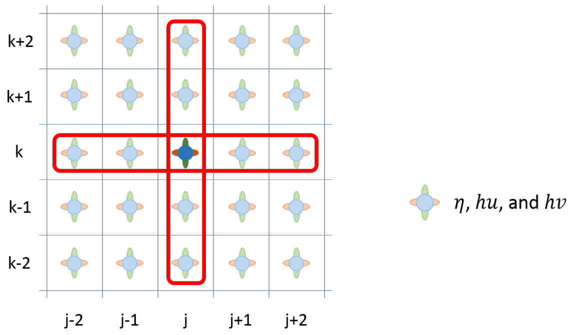


Figure 5: Stencil for the KP and CDKLM schemes. Blue circles represent η, while orange horizontal ellipses and green vertical ellipses represent U and V, respectively.

Here, Δt, is restricted by a CFL-condition which ensures that disturbances travel at most one quarter of a grid cell per time-step,

$$\Delta t \leq \frac{1}{4}\min\left\{\frac{\Delta x}{\max_\Omega\left|u \pm \sqrt{g(H+\eta)}\right|}, \frac{\Delta y}{\max_\Omega\left|v \pm \sqrt{g(H+\eta)}\right|}\right\}. \tag{14}$$

The results from (13) are again *averaged* cell values at time t_{n+1}. In general, this is called a REA-algorithm [6], named after the steps reconstruct, evolve, and average. The KP scheme was originally designed to solve the non-rotating shallow-water equations. It is therefore worth noting that the discretization of the Coriolis source term here is made naïvely, as

$$S_f(\boldsymbol{Q}_{j,k}) = [0, f(hv)_{j,k}, -f(hu)_{j,k}]^T.$$

Further details can be found in Kurganov and Petrova [2], and a GPU implementation is presented by Brodtkorb et al. [3].

The stencil for the KP scheme is shown in Figure 5. Note that no variables are defined on the exact domain boundary. Instead, boundary conditions on all three conserved variables are required to be defined in two layers of ghost cells surrounding the computational domain. To start the simulation, initial conditions must be defined at $t = t_0$.

3.4 The CDKLM Scheme

The fourth scheme considered in this paper is a novel high-resolution finite-volume scheme, presented by Chertock et al. [4] (CDKLM). It is similar to the KP scheme with respect to the numerical formulation of flux terms, and is based on the same semi-discrete form as in (12).

Whereas KP is designed to be well-balanced with respect to the steady-state, lake-at-rest solution, the CDKLM scheme aims to be well-balanced with respect to the steady-state solution given by the geostrophic balance, from (33). This is achieved by introducing reconstruction variables $\boldsymbol{R} = [u, v, K, L]^T$. Here, K and L are Coriolis potentials given by

$$K := g(\eta - f_v), \qquad L := g(\eta + f_u), \tag{15}$$

in which $[f_u, f_v]^T$ are the primitives of the Coriolis force, defined through their derivatives,

$$(f_v)_x := \frac{f}{g}v, \qquad (f_u)_y := \frac{f}{g}u. \tag{16}$$

The CDKLM scheme follows a similar REA-algorithm as the KP scheme, but instead of basing the reconstruction on $\boldsymbol{Q}$, the scheme reconstructs a piecewise bilinear polynomial of $\boldsymbol{R}$. Piecewise reconstructions of η, u, and v are then obtained from the reconstructed $\boldsymbol{R}$, and used to evaluate central-upwind numerical fluxes. These are used to update $\boldsymbol{Q}$ in time through a sufficiently accurate total-variation-diminishing Runge–Kutta method.

In summary, the CDKLM scheme uses the same numerical flux function and reconstruction methodology as the KP scheme, but takes the Coriolis forces into account for properly representing known steady-state solutions. Both these schemes are considered to investigate whether the more complex handling of the Coriolis forces pays of in terms of better simulation results. The shape of the stencil for the CDKLM scheme is similar to the one for KP, shown in Figure 5. Because of this, the requirement to initial and boundary conditions are the same for the two schemes, as well.

3.5 Boundary Conditions

All schemes in the simulation framework support three different boundary conditions: reflective wall-, periodic-, and open-boundary conditions. All three are implemented by modifying the solution in the ghost cell region, which constitute of a band of cells or grid points encircling the original computational domain. As a minimum, the ghost cell region is required to be large enough so that the outermost grid points in the original domain can be computed with the chosen stencil. In general, boundary conditions are more conveniently implemented using staggered grids where the grid points for the velocity components and the pressure (or free surface) do not coincide. Staggered grids are also very useful to maintain the geostrophic balance, so that difference schemes for the components of the pressure gradient are centered at the relevant (u, v) velocity grid points.

The reflective wall boundary condition is designed to preserve mass and momentum within the original domain. This is achieved by creating an ingoing momentum in the ghost region equal to the outgoing momentum in the interior. Let x^W be the location of the western boundary, so that $x = x^W + \hat{x}$ is in the interior of the domain for some $\hat{x} > 0$. Reflective wall boundary conditions are obtained by setting the values in the ghost region according to

$$\begin{aligned} \eta(x^W - \hat{x}) &= \eta(x^W + \hat{x}), \\ hu(x^W - \hat{x}) &= -hu(x^W + \hat{x}), \\ hv(x^W - \hat{x}) &= hv(x^W + \hat{x}). \end{aligned} \tag{17}$$

Note that the condition on hu implies that $hu(x^W) = 0$, so that no water is allowed to leave the domain. The same principle applies in the y-direction.

Periodic boundary conditions make flow reaching one boundary reappear on the opposite boundary. Let ϕ be any of the conserved variables, and let x^W and x^E be the location of the western and eastern boundary, respectively. Periodic boundary conditions are then defined as

$$\phi(x^W - \hat{x}) = \phi(x^E - \hat{x}), \tag{18}$$

for $\hat{x} > 0$, so that $x^W - \hat{x}$ is in the ghost region. Note that for staggered grids, discharge on the boundary needs to be computed by the stencil, expanding the computational domain with one extra row and column for v and u, respectively.

Regional ocean circulation models that cover limited areas require open boundary conditions in which external solutions can be imposed in such a way that external signals can propagate freely into the model domain and signals generated in the interior can propagate freely out of the domain [40]. To maintain the explicit and parallel structure of the simulation code, a flow relaxation scheme [41, 42] is used. Let ψ^{ext} be the exterior solution far away from the interior domain. Define a ghost cell region consisting of N_G ghost cells, where N_G is larger than required by the shape of the numerical stencil. At each time-step, the stencil is applied to all cells having a sufficient number of neighbouring cells. The flow relaxation scheme relaxes the obtained solution in all ghost cells towards ψ^{ext},

$$\psi = (1 - \alpha)\psi^* + \alpha\psi^{ext}. \tag{19}$$

Here, ψ^* is the solution from the numerical stencil, and α is a relaxation factor, so that $\alpha = 0$ at the boundary and $\alpha = 1$ in the outermost ghost cell. In this paper, the size of the ghost cell region is chosen to be $N_G = 10$ for all schemes, along with relaxation parameter

$$\alpha = 1 - \tanh\left(\frac{1 - i}{3}\right), \tag{20}$$

for $i = 0, ..., N_G - 1$.

Table 1: Comparison of properties of the four numerical schemes. The ghost cell region corresponds to how many rows of ghost cells for η you need to implement wall boundary conditions. The initial conditions row shows that CTCS needs the conserved variables for two time-steps.

	Numerical Scheme			
	FBL	**CTCS**	**KP**	**CDKLM**
Underlying equation	Linearized	Nonlinear	Nonlinear	Nonlinear
Coriolis discretization	Intrinsic	Intrinsic	Naïve addition	Intrinsic
Conservative	Yes	Yes	Yes	Yes
Discretization method	Finite difference	Finite difference	Finite Volume	Finite Volume
Grid	Arakawa C	Arakawa C	Arakawa A	Arakawa A
Order in space	2	2	2	2
Order in time	1	2	2	2
Ghost cell region	0	1	2	2
Initial conditions	1 (t_0)	2 (t_0, t_{-1})	1 (t_0)	1 (t_0)
CFL-condition relative to $\Delta x/c_0$	$1/\sqrt{2}$	$1/(2\sqrt{2})$	$1/4$	$1/4$
Relative numerical complexity	Simple	Fairly simple	Fairly complex	Complex

3.6 Discussion

Table 1 gives a structured comparison of the four numerical schemes discussed above. The scheme that stands out the most is FBL, as it only solves the linearized equations, but it is still of interest due to its simplicity, and thereby its computational efficiency. The KP scheme is also notably different from the others, with its naïve discretization of the Coriolis forces. It is included in this study as it is interesting to investigate the differences between a scheme that is well-balanced with respect to lake-at-rest (KP), and a slightly more computational intensive scheme that is also well-balanced with respect to the geostrophic balance (CDKLM). All three nonlinear schemes are of second order in both time and space, while FBL is only first order in time.

On the practical side, there is a difference in the data dependency for the four schemes. We see that the finite-difference schemes have a smaller ghost cell region than the finite-volume schemes, and a consequence of this is that the boundary conditions are easier to implement for CTCS and FBL compared to for KP and CDKLM. Also, since the finite-difference schemes use an Arakawa C grid, it is easier to prescribe the momentum on the boundary than for the finite volume schemes which use an A grid. On the other hand, CTCS requires the state of the two previous time-steps in order to evolve, whereas the other three schemes require only one. A final advantage for the finite-difference schemes is that they allow for a larger time-step compared to the finite-volume schemes.

4 Test Cases and Results

In this section we evaluate the quality and behavior of the four numerical schemes described in Section 3 through a set of six test cases. The test cases focus on the intrinsic dynamics of shallow-water flows in a rotating reference frame, and we ignore the impact of atmospheric pressure, and friction caused by bed and wind shear stresses. It should be noted that although we study barotropic dynamics here, the equations are formally identical to the ones in so-called 1.5-layer models for a well mixed upper ocean layer above a deep ocean layer at rest [8]. A 1.5-layer model setup would in some cases provide for more geophysically relevant examples, e.g. Cases B and C, which in their present implementation have unrealistically large domains. We begin with non-rotating properties of the schemes in Case A, before we study the transient development arising from an initial disturbance in the sea surface elevation, investigating the so-called geostrophic (or Rossby) adjustment problem [43] in Cases B and C. We then look at Kelvin waves in Case D, in which fast moving shallow-water waves are trapped along the coast by the Coriolis force. The final two test cases, E and F, focus on Rossby waves caused by the dynamics in the potential vorticity, driven by variation in the Coriolis force and in the bathymetry, respectively. The physics behind the oceanographic test cases are described in more detail in A, but the essential points are summarized for each case for convenience.

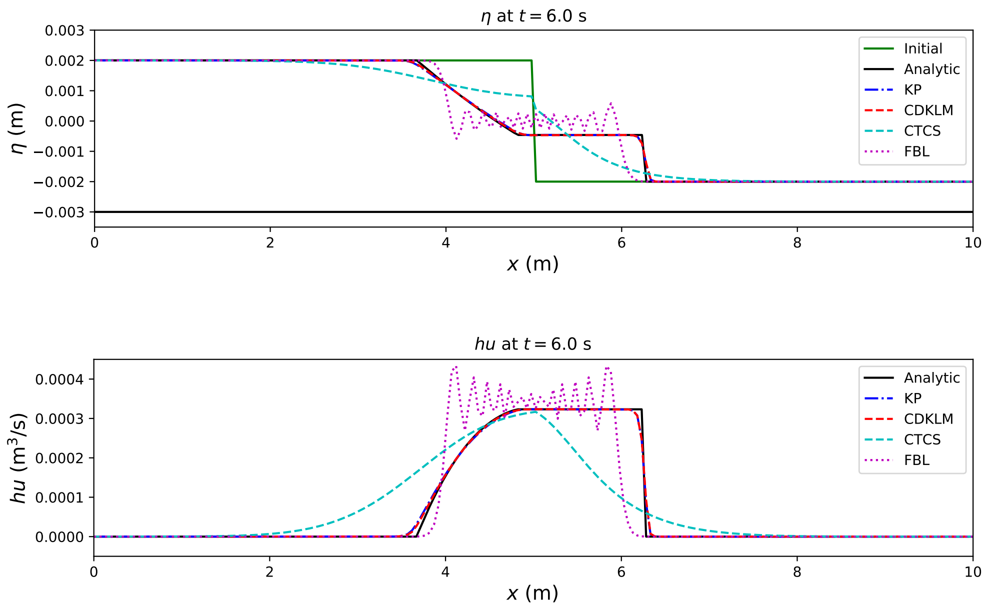


Figure 6: Case A: SWASHES test case 4.1.1 "dam break on a wet domain", comparing the result from four different numerical schemes to the analytic solutions. Whereas the KP and CDKLM schemes are able to capture the shock solution, FBL becomes unstable, and the CTCS scheme gives different results for different choices for A. Here we have used $A = 0.1$, but no values of A give the analytic results.

4.1 Case A: Traditional Non-Rotating Shallow-Water Benchmarks

Our first benchmark case is described in SWASHES [21] and is a classical dam break case for non-rotational physics. The first part is "lake-at-rest with an immersed bump" [21, Sec. 3.1.1], which validates that the numerical scheme is able to preserve a calm lake on top of a non-constant bathymetry without any initial momentum. All four schemes pass this test.

The second part is a one-dimensional "dam break on a wet domain without friction" [21, Sec. 4.1.1], which validates the scheme's ability to capture shocks. The initial conditions consist of a step function in η with zero momentum, shown along with the simulation results in Figure 6. The analytic solutions [21] are given as black solid lines in the plot. This test case clearly shows that the finite-volume schemes are able to capture the shock with high precision, whereas the finite-difference schemes fail. The FBL scheme results in unstable oscillations, while CTCS gives various incorrect results for different choices for the eddy viscosity parameter A. High values for A gives a solution that resembles the initial state, and very low values for A results in unstable solutions. This is expected, given that FBL and CTCS with $A = 0$ have no numerical diffusion, causing the schemes to break down in the presence of shock solutions. KP and CDKLM, on the other hand, have built-in numerical diffusion in their numerical fluxes, and are designed to capture discontinuities. Oceanographers are often not too concerned with shock solutions, as these are not part of the most important physics of oceanography, and this test illustrates that the finite difference schemes considered here are not designed with shocks in mind.

4.2 Case B: Rossby Adjustment

In this test case, and in Case C, we look at Rossby adjustment, a smooth dam break problem with rotation, in which an initial surface elevation in a limited area adjusts itself under gravity towards equilibrium. The case is designed to

Table 2: Parameters used in the oceanographic test cases. Here, (N_x, N_y) denote the number of cells or grid points within the grid, and $(\Delta x, \Delta y)$ denotes the size of each cell or distance between grid points. The interesting features of each of the cases are centered in (x_0, y_0). The surface elevation, and volume transport in x- and y-direction are given by η, hu, and hv, respectively, with initial conditions given at time t_0. The solution is developed forward in time with time-step Δt until time T. The other parameters describe ocean depth (H), gravitational force (g), Coriolis forcing (f), and eddy viscosity parameter A for CTCS. Where applicable, f_0 and β are parameters in the Coriolis beta-plane model, and α describes the slope of the ocean bed. We have used the same time-step size for all schemes.

	Test cases			
	B and C: Rossby adjustment	**D: Kelvin waves**	**E: Planetary Rossby waves**	**F: Topographic Rossby waves**
(N_x, N_y)	$(800, 1000)$	$(1000, 200)$	$(400, 400)$	$(400, 400)$
$(\Delta x, \Delta y)$	$(50\text{ km}, 50\text{ km})$	$(5\text{ km}, 10\text{ km})$	$(20\text{ km}, 20\text{ km})$	$(20\text{ km}, 20\text{ km})$
(x_0, y_0)	$(20\ 000\text{ km}, 25\ 000\text{ km})$	$(2\ 502.5\text{ km}, -5\text{ km})$	$(6\ 400\text{ km}, 4\ 000\text{ km})$	$(6\ 400\text{ km}, 4\ 000\text{ km})$
Δt	100 s	25 s	100 s	100 s
$\eta(x, y, t_0)$	Eq. (21)	Eq. (23) with $\eta_0 = \{0.05\text{ m}, 2.0\text{ m}\}$	Eq. (27)	Eq. (27)
$hu(x, y, t_0)$	0	Eq. (25)	Eq. (28)	Eq. (28)
$hv(x, y, t_0)$	0	0	Eq. (29)	Eq. (29)
$H(x, y)$	1000 m	100 m	25 m	Eq. (30)
g	9.81 m/s^2	9.81 m/s^2	9.81 m/s^2	9.81 m/s^2
f	$1.2 \cdot 10^{-4}\text{ s}^{-1}$	$1.2 \cdot 10^{-4}\text{ s}^{-1}$	Eq. (26)	$8.0 \cdot 10^{-4}\text{ s}^{-1}$
f_0	-	-	$8.0 \cdot 10^{-4}\text{ s}^{-1}$	-
β	-	-	$2.0 \cdot 10^{-11}\ (\text{ms})^{-1}$	-
α	-	-	-	$2.5 \cdot 10^{-7}\text{ m}^{-1}$
A	0	25	0	0
T	$1.257 \cdot 10^{7}$ s	$10\frac{\Delta x N_x}{\sqrt{gH}} \approx 1.6 \cdot 10^{6}$ s	$6.0 \cdot 10^{6}$ s	$6.0 \cdot 10^{6}$ s
Boundary conditions	Flow relaxation scheme, Eq. (19) and (20)	North/South: Wall, Eq. (17) East/West: Periodic, Eq. (18)	Flow relaxation scheme, Eq. (19) and (20)	Flow relaxation scheme, Eq. (19) and (20)

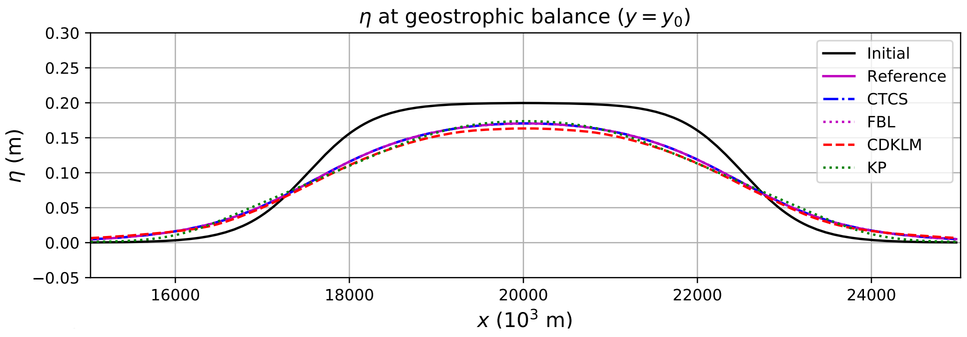


Figure 7: Case B: Steady-state results for Rossby adjustment on a flat bathymetry. The initial condition for η is shown in black. All simulators give results close to the reference solution, with slightly better results for FBL and CTCS than for KP and CDKLM.

trigger a gravitational wave moving radially outwards from the center, similar to a radial dam break. Due to the rotation, parts of the fluid will become trapped within a length scale of the Rossby radius $L_R = c/f$ from the initial disturbance. Here, $c = \sqrt{gH}$ is the phase speed of free gravitational waves, and the basic balance will be the geostrophic solutions (33). In this case, f is considered to be constant, whereas variations in rotation and topography will be considered in cases E and F.

The Rossby adjustment problem investigates a scheme's ability to obtain and preserve steady-state solutions given by the geostrophic balance (33). We have a constant bathymetry, $H(x, y) = H_0$, and the initial state of the surface is formed as a bump,

$$\eta(x, y, t_0) = \frac{1}{2}\eta_0 \left[1 + \tanh\left(\frac{-\sqrt{(x - x_0)^2 + (y - y_0)^2} + D}{L}\right)\right], \tag{21}$$

in which D, L, and η_0 are related to the width, steepness, and maximum height of the bump, respectively. The center of the bump is positioned at (x_0, y_0), and the initial momentum is zero,

$$hu(x, y, t_0) = hv(x, y, t_0) = 0. \tag{22}$$

The boundary conditions are set to allow gravity waves to leave the domain according to (19) and (20), and the solution should converge to a steady state consisting of a wide rotating bump (satisfying the Klein-Gordon equation (67) in A).

The domain consists of 800×1000 cells with $\Delta x = \Delta y = 50$ km. The depth is chosen to be $H = 1000$ m, and the full set of parameters are listed in Table 2, along with $D = 50\Delta x$, $L = 15\Delta x$ and $\eta_0 = 0.2$ m for (21). The steady-state solutions along $y = y_0$ at time T are shown in Figure 7. The reference solution is obtained by solving the Klein-Gordon equation (67) by a finite-difference approach, with the given initial condition and $\eta = 0$ along the boundaries. All four schemes qualitatively capture the same steady-state solution defined through the geostrophic balance.

The relative discrepancy between the simulated results and the solution of the Klein-Gordon equation, is shown in Figure 8. The finite-volume schemes are off by up to 4%, and the staggered finite-difference schemes are within machine precision of the reference.

Figure 9 explores the radial symmetry of the steady-state solutions. The dense collection of bright yellow dots represent values of η at grid points in a straight eastward line from (x_0, y_0), whereas darker blue dots represent all other grid point values. The x-axis represents the Euclidean distance from (x_0, y_0). Perfect radial symmetry would result in all the blue dots disappearing behind the yellow dots. Small deviations are present for all schemes, with CTCS and FBL being slightly better than CDKLM and KP. It should also be noted that we have observed small numerical oscillations in the steady-state solution produced by CDKLM, which we have not been able to reproduce for the other non-steady-state test cases.

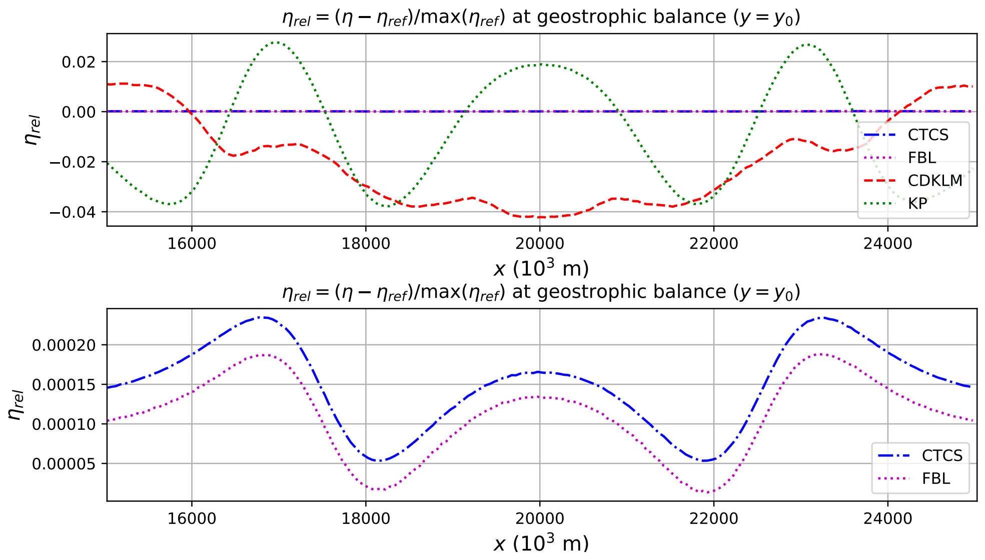


Figure 8: Case B: Discrepancy between the obtained steady-state solutions and the reference solution for the Rossby adjustment problem. The KP and CDKLM schemes obtain a steady state with a relative error of 4% compared with the reference solution. The CTCS and FBL schemes capture the reference solution with machine precision.

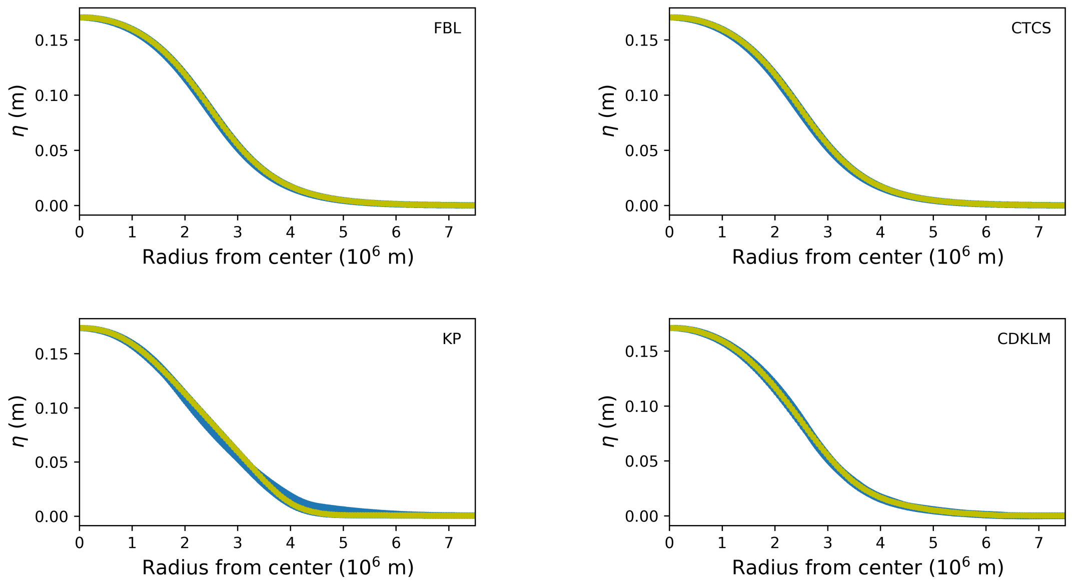


Figure 9: Case B: Radial symmetry at geostrophic balance, obtained from the Rossby adjustment problem on flat bathymetry. The bright yellow line is the η-values obtained along a straight line eastwards from the center of the domain, while the darker blue area consists of a scatter plot of all η-values radially distributed from the same center point. Deviations of the blue scatter from the yellow line indicate radial asymmetry. It is therefore clear that FBL and CTCS are slightly better than KP and CDKLM at preserving radial symmetry.

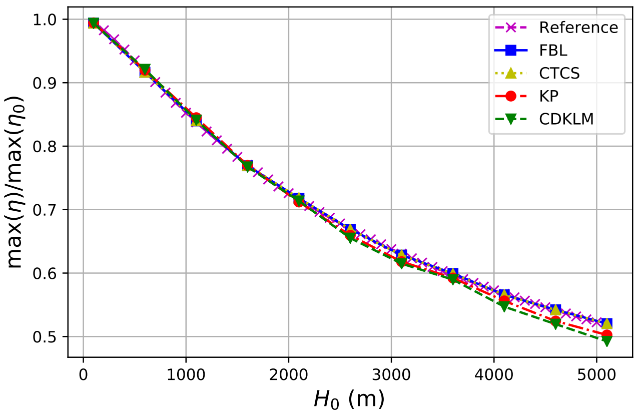


Figure 10: Case C: Relative height of the Rossby adjustment steady state η obtained by the different schemes for different depths, compared with a reference solution. The FBL and CTCS schemes give good match, whereas the KP and CDKLM schemes have slightly too low steady state bumps for large depths.

4.3 Case C: The Adjustment Problem for Variable Rossby Radius of Deformation

This case checks the schemes ability to capture how the Rossby radius of deformation in Case B depends on water depth. We vary H_0 from 100 to 5000 m, and study the ratio between the maximum steady state surface elevation and the maximum initial surface elevation. Initial and boundary conditions are the same as for Case B.

The phase speed c of free gravity waves increases with depth, and hence the Rossby radius $L_R = c/f$ also increases. The relevant parameter here is the ratio between the horizontal scale of the initial surface elevation and the Rossby radius. With increasing depth, the Rossby radius increases compared with the fixed initial disturbance. This means that more potential energy is converted to kinetic energy in the form of gravitational waves that propagate away. The geostrophically balanced steady-state solution therefore becomes distributed over a larger area. We would expect that the maximum surface elevation in the steady-state solution decreases as the depth and the Rossby radius increases. Such reduction is exactly what is seen when calculating the reference solution for different water depths using the same method as in Case B.

Figure 10 shows the experimental results for all four numerical schemes. The FBL and CTCS schemes result in almost identical steady-state solutions for all values of H_0, with a very good match to the reference solution. The finite-volume methods on the other hand, give slightly lower steady-state solutions for the surface elevation for the deep cases. This difference indicates that the finite-volume methods may be more diffusive. We will discuss this possibility in relation to Cases E and F.

4.4 Case D: Kelvin Waves

Kelvin waves are shallow-water waves trapped along the coast that propagate with the cost on their right / left side on the the northern / southern hemisphere. For this case, we consider a rectangular domain with constant depth, wall boundary conditions, (17), along the southern and northern boundaries, and a periodic boundary condition, (18), over the east-west boundary. The initial state consists of a wave trapped at the southern boundary and travelling eastwards. The initial surface elevation is

$$\eta(x, y, t_0) = \frac{\eta_0}{2} \exp\left(\frac{-\sqrt{(y - y_0)^2}}{L_R}\right) \left[1.0 + \tanh\left(\frac{-\sqrt{(x - x_0)^2} + L_R}{L_R/3}\right)\right], \tag{23}$$

and the momentum is based on the geostrophic balance, (33), expressed with respect to the conserved variables as

$$hu(x,y) = -\frac{gH}{f}\frac{\partial \eta}{\partial y} \quad \text{and} \quad hv(x,y) = \frac{gH}{f}\frac{\partial \eta}{\partial x}. \tag{24}$$

In the case of Kelvin waves, the momentum is balanced according to (24) along the barrier, where hv is set to zero. Using (23), we get that

$$hu(x,y,t_0) = \sqrt{gH} \cdot \operatorname{sign}(y - y_0)\eta(x,y,t_0), \tag{25}$$

$hv(x,y,t_0) = 0$. We let $(x_0, y_0) = (2\ 502.5\ \text{km}, -5\ \text{km})$, which is chosen so that the maximum height of the wave is in a cell center in the middle of the domain in x-direction, and the wave height is still increasing in the y-direction as it meets the southern boundary. The amplitude of the wave is given by η_0, and we will use two different values for η_0 in this test case. All other parameters are listed in Table 2.

From linear wave theory, as discussed in A, Kelvin waves are standing waves travelling with speed $\sqrt{gH}$ along the coast. By selecting a small amplitude, $\eta_0 = 0.05$ m, we expect approximately linear behavior even from the nonlinear equations, and the period in our domain becomes $T = \Delta x N_x / \sqrt{gH}$. Figure 11 shows the resulting Kelvin waves for all four schemes as cross sections along the x-axis, focusing on the middle section of the domain after one, five and ten periods. The solid lines represent $\eta(x, \frac{1}{2}\Delta y, t)$ along the southern boundary and the dotted lines show $\eta(x, (30 + \frac{1}{2})\Delta y, t)$ some distance away from the boundary. The waves have momentum towards the right. We see that FBL, CTCS and CDKLM give close linear advection as expected. If we look carefully, however, we note that there are tendencies of oscillation at $t = 10T$, with a wave building up on the front of the wave, and a depression just behind the wave. This is a typical artifact of such schemes [44], and is exaggerated for coarser resolutions. The KP scheme also maintains the periodicity in the solution, but the amplitude of the Kelvin wave is decreasing, and an edge builds up behind the wave. This is most likely caused by the naïve discretization of the Coriolis forcing term.

Next, we look at Kelvin waves with a larger amplitude by setting $\eta_0 = 2.0$ m, and these results are shown in Figure 12. As expected, the FBL scheme produces the same periodic solution as for the small amplitude wave, as it solves the linearized equations. The other three schemes solve the nonlinear equations where the wave speed is a function of the full depth $H + \eta$ (meaning that the particle velocity is expected to larger on the crest of the wave), and we expect that the wave should sharpen towards a shock with time. This is the behavior observed by the KP and CDKLM schemes, shown in the two lower panels of Figure 12, and the same tendency can also be observed for CDKLM in Figure 11. CTCS, on the other side, is not able to produce this solution, and instead produces a solution with the opposite behavior, as seen in the second panel of Figure 12. Here, the crest of the wave is travelling slower due to numerical dispersion, which is a known property of the CTCS scheme [45, 8]. The consequence is that the phase speed slows down, and generates a shock behind the wave. We have been able to reproduce this behaviour for a wide range of values for the eddy viscosity parameter, A, and since CTCS is unable to maintain shock solutions, the shock results in spurious oscillations.

In total, we see that the FBL, KP and CDKLM schemes behave as expected from a physical point of view, and CTCS behaves as expected from a numerical point of view. The finite-volume methods are better at maintaining fast-travelling waves than the CTCS scheme. It should also be noted that the behavior from the results in Figure 12 can be reproduced for pure gravity waves in non-rotating domains as well.

4.5 Case E: Planetary Rossby Waves

The two last test cases focus on dynamics related to potential vorticity, which is defined as the total vertical vorticity divided by the total water depth. The total vorticity can be split into two parts: one dominating planetary part caused by Earth's rotation and a relative part caused by the local vorticity of the flow in the rotating reference frame. The potential vorticity expresses the angular momentum of a material water column, and its conservation causes the spin of a water column to increase if the column is stretched vertically and vice versa. Large-scale changes in the relative vorticity are due to horizontal wind shear and are important for the basin-scale circulation in the oceans. Changes in the relative vorticity can also occur on smaller scales when a column of water is advected into deeper or shallower areas. Changes in the planetary vorticity occurs when a water column is advected either northwards or southwards because the vertical component of the Earth's rotation (the Coriolis parameter f) is different. In non-dissipative systems, and in the absence of atmospheric forcing, the potential vorticity is conserved.

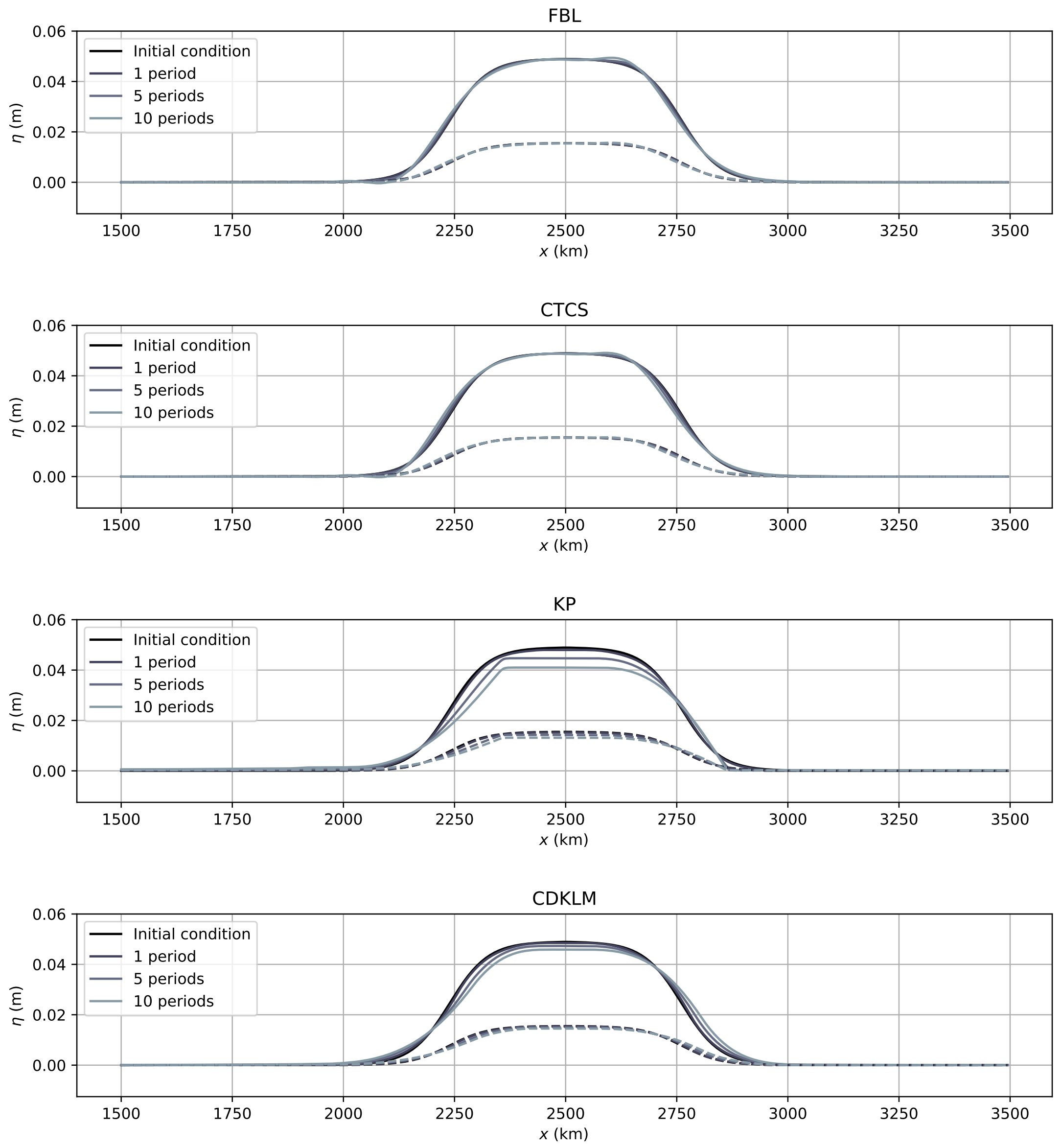


Figure 11: Case D with small waves: Small amplitude Kelvin waves travelling along the southern wall boundary, in a east-west periodic domain. The solid lines represent values of η along the boundary, while the dotted lines show η 30 grid cells into the domain. The initial state and the solution after one, five and ten periods are plotted. The small amplitude of the wave gives approximately linear behavior for all four schemes, but we see that the wave produced by the KP scheme is decreasing with an unnatural edge at the back of the wave.

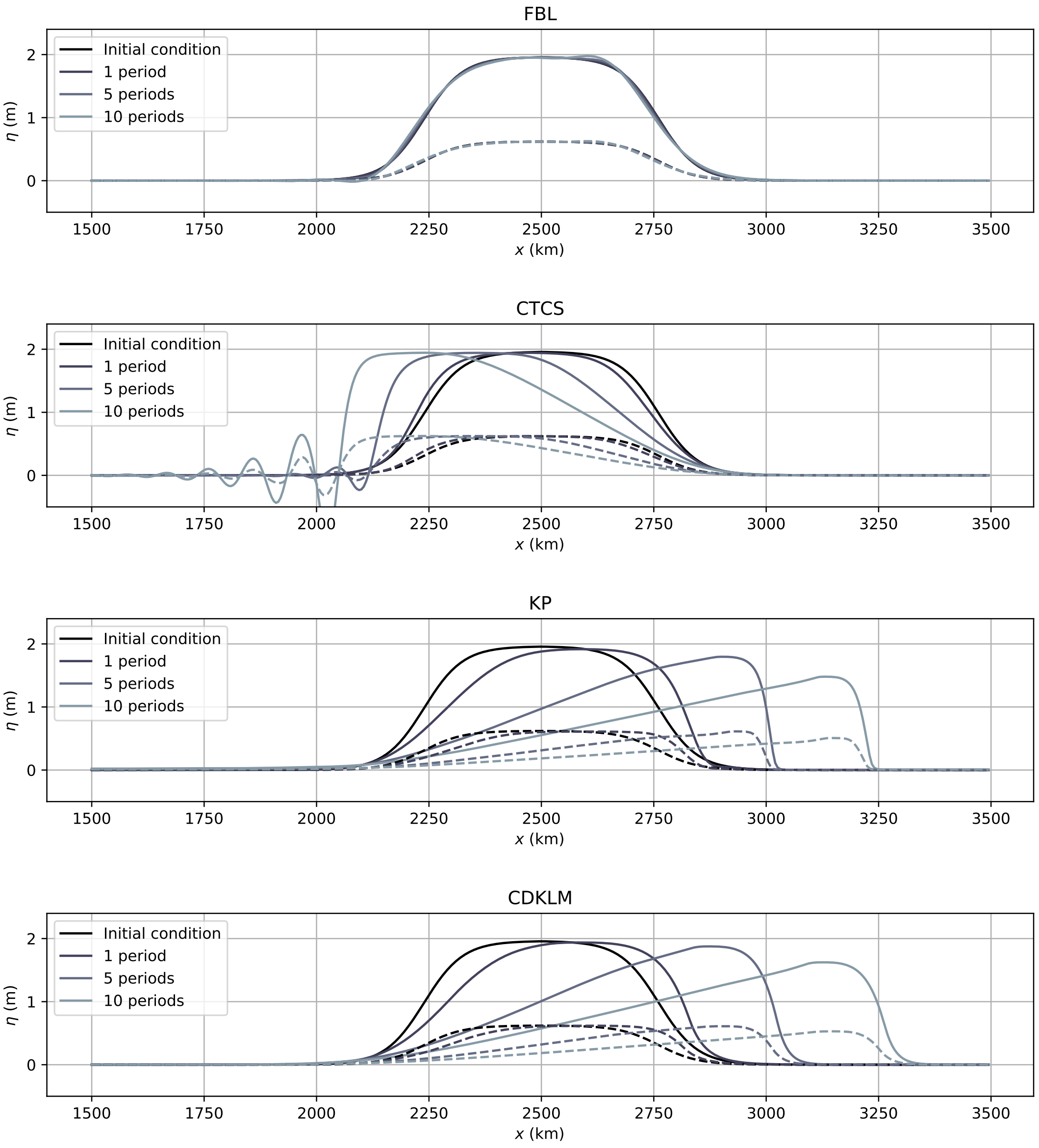


Figure 12: Case D with large waves: Kelvin waves travelling along the southern wall boundary, in a east-west periodic domain. The solid lines represent values of η along the boundary, while the dotted lines show η 30 grid cells into the domain. The initial state and the solution after one, five and ten periods are plotted. The figure shows that FBL gives results according to linear wave theory. In the nonlinear equations, the top of the wave is expected to have a higher phase speed than the slopes of the wave, causing a shock to be slowly generated. This behavior is well captured by the KP and CDKLM schemes, but the CTCS scheme is slowed down due to numerical dispersion.

Disturbances in the vorticity may induce a class of waves referred to as Rossby waves. If changes in the vorticity are associated with varying bathymetry, we obtain so-called topographic Rossby waves that propagate along lines of constant depth. Alternatively, we can obtain planetary Rossby waves for which gradients in the local rotation rate f determine the propagation characteristics. These waves have a primary component in the zonal direction, and a brief description of the relevant (linearized) theory is provided in A.

Planetary Rossby waves depend on variations in the local rotation rate f. In these test cases, we consider a beta-plane model of the Coriolis force, given by

$$f(y) = f_0 + \beta (y - y_0). \tag{26}$$

Here, f_0 is the Coriolis force at y_0, defined as $f_0 = 2\omega \sin(\phi_0)$, in which $\omega = 2\pi/(24\,\mathrm{hours})$ is the angular velocity of the Earth and ϕ_0 is the latitude at y_0. The parameter β is hence

$$\beta = \frac{\partial f}{\partial y} = \frac{2\omega}{R}\cos(\phi_0),$$

in which R is the radius of the Earth.

In the following test case, y_0 is chosen at approximately 33° north in the center of the domain, giving the values for f_0 and β presented in Table 2. The domain is chosen to consist of 400×400 cells with $\Delta x = \Delta y = 20$ km, with a depth of $H = 25$ m. The initial disturbance is a Gaussian bump,

$$\eta(x, y, t_0) = \eta_0 \exp\left\{-\frac{(x - x_0)^2 + (y - y_0)^2}{r_0}\right\}, \tag{27}$$

with $\eta_0 = 0.25$ m, $(x_0, y_0) = (0.8\Delta x N_x, 0.5\Delta y N_y)$, and $r_0 = 2.5 \cdot 10^6$ m^2. To avoid that most of the potential energy from the bump results in gravity waves, the initial momentum is constructed by inserting (27) into the geostrophic balance (24). The initial momentum then becomes

$$hu(x, y, t_0) = 2\frac{gH(y - y_0)}{f_0 r_0}\eta(x, y, t_0), \tag{28}$$

and

$$hv(x, y, t_0) = -2\frac{gH(x - x_0)}{f_0 r_0}\eta(x, y, t_0). \tag{29}$$

Flow relaxation boundary conditions are used, and all parameters are summarized in Table 2.

Simulation results of planetary Rossby waves from all four schemes are shown in Figure 13. The right-hand panels show the final state of η at time $t = 6.0 \cdot 10^6$ s, corresponding to approximately 70 days (60 000 time-steps). From the initial disturbance, we can see the Rossby waves propagating westwards, with significant components in the north-south direction. All schemes tend to generate Rossby waves with larger amplitudes to the north than to the south, where the Coriolis force is weaker. The left-hand side panels in Figure 13 are Hovmöller diagrams, which illustrate how the waves develop and propagate along the x-axis at $y = y_0$ over time. From these diagrams we may note that the Rossby waves propagate from east to west as theory predicts. This can be seen from the lines of constant phase that extend upwards to the left. We also see that there are two dominating modes with different phase speeds, and that the wave energy slowly propagates eastwards; that is, the x-components of the group and phase velocities have different signs.

Because the test case is two-dimensional, there are no simple analytical solutions to use as reference. We would, however, expect the CTCS scheme to perform quite well, since this scheme is specifically tailored for problems such as the one considered here. We note in particular that the KP scheme has a much higher dissipation rate than the other schemes. This may be caused by the naïve discretization of the Coriolis source terms in the KP scheme.

4.6 Case F: Topographic Rossby Waves

The wave phenomena shown in Case E can also be produced by varying the bottom topography rather than the Coriolis force, and is then called topographic Rossby waves. We have here chosen parameter values so that the topographic

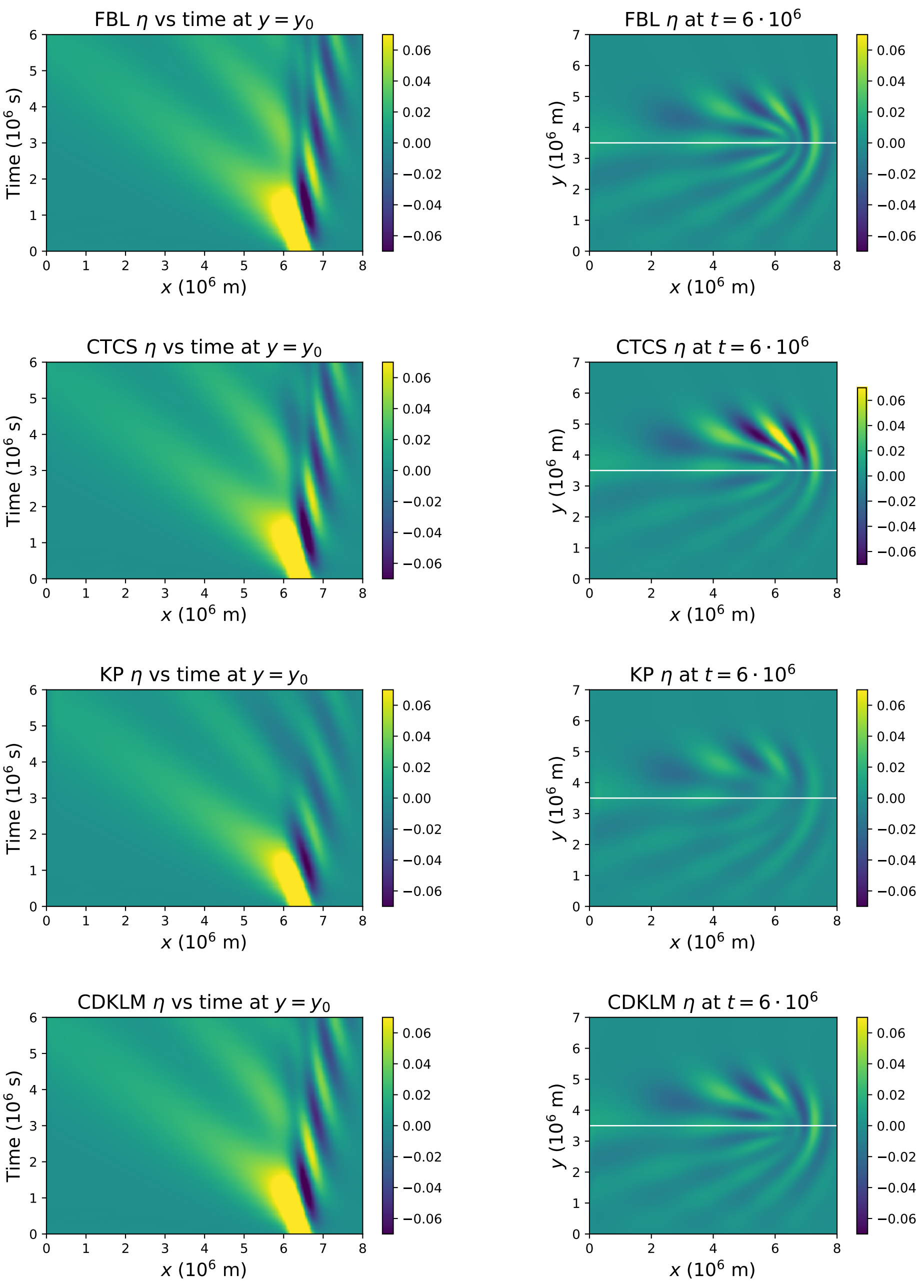


Figure 13: Case E: Long-term simulation results when investigating planetary Rossby waves caused by a linear Coriolis parameter. The left-hand figures show Hovmöller diagrams located along y_0. The figures at the right-hand side show η after the last time-step, with the cross section at y_0 illustrated with a white line. Note that KP has a more dissipative solution than the others, as the eastwards energy is lost in this scheme. Note also that the solutions are not symmetric about y_0.

Rossby waves should (roughly) correspond to the planetary Rossby waves considered above. The bathymetry is modelled as a linear function of the form

$$H(x, y) = H_0 \left[1 - \alpha(y - y_0)\right]. \tag{30}$$

Here, y_0 and H_0 are chosen to be the same as for the case with planetary Rossby waves, and we use constant $f(y) = f_0$. The slope parameter for the bottom, α, is chosen according to the Coriolis force in the above case, so that $\alpha = \beta/f_0$ (see A for details). The domain, and boundary and initial conditions are chosen to be the same as in the planetary case.

Simulation results for the topographic Rossby waves are shown in Figure 14, with Hovmöller diagrams on the left-hand side and η after the last time-step on the right-hand side. Again, there is no simple analytical solution to use as reference, and we cannot expect the topographic and planetary Rossby wave cases to be identical. The solutions are qualitatively similar, with westward propagating waves (two main modes) and eastward propagating wave energy, but the solutions are more symmetric across the centerline $y = y_0$.

The three schemes FBL, CTCS, and CDKLM give results that are quite similar to each other, while the KP scheme is dissipative in the same way as for the planetary Rossby waves. The overall amplitudes are somewhat smaller for CDKLM than for the two staggered schemes.

4.7 Numerical Order and Performance

We have determined the numerical order of convergence for the four different numerical schemes. The test case is a smooth problem that avoids the formation of shocks during the simulation period and consists of an initial radial cosine bump at the center of the domain with zero momentum and wall boundary conditions. The domain measures 512×512 km, with a gravitational coefficient of $9.81\ \mathrm{m/s^2}$, a Coriolis coefficient of 0, and constant depth $H_0 = 50$ m. The radial cosine bump is defined as

$$\eta_{j,k} = \begin{cases} \frac{1}{2}\eta_{\max}\left(1 + \cos\left(\frac{r}{c}\pi\right)\right), & \forall\, r \le c \\ 0, & \forall\, r > c, \end{cases}$$

in which $\eta_{\max} = 1.0\ \mathrm{cm}$, r is the distance from the center of the domain, and c is the radial size of the bump, in our case 60% of the domain width. For the finite-volume schemes, we start by generating a fine scale initial η, and integrate the variables over the coarse grid cells to make sure we solve the exact same initial conditions. For the finite-difference schemes we simply evaluate the point values on the cosine bump. We then simulate until $t = 30$ minutes, and compute the difference between the fine scale reference and the coarse scale simulation. The reference is generated by simulating on a fine mesh, and the fine-scale solution is then coarsened using the same method as for the initial conditions. The error norms have been computed in double precision.

The results are shown in Figure 15, showing second-order convergence for all schemes except FBL, as expected. It should be noted that CTCS is a leapfrog scheme that is only second order accurate every other time-step (see e.g., [8]).

Practical oceanographic simulation is demanding on computational capacity, and it is therefore important that the numerical schemes allow for fast and efficient implementations, especially for ensemble-based simulation. It has previously been shown that explicit finite-difference and finite-volume schemes, such as those considered herein, have an inherent parallelism that can readily be exploited in many-core computing [13, 14, 15, 3, 16]. The actual performance of each scheme will in general vary upon the specific simulation case, domain size, hardware, and level of optimization, and we therefore choose to normalize all performance values with respect to the fastest non-linear scheme (CTCS). To evaluate the computational performance we run the above case on a grid consisting of 2048×2048 cells for $t = 24$ hours, using a time-step that incorporates each scheme's CFL condition respectively, with a Courant number of 0.9. All the schemes have been been optimized to a comparable level, and are run with optimal configurations [46]. By comparing the wall clock time for each scheme, we see that CTCS completes the simulation roughly $4.3\times$ and $5.7\times$ faster than KP and CDKLM, respectively. By looking at iterations per second, we see that KP and CDKLM requires $3\times$ and $4\times$ as much time for each time-step, respectively, compared to CTCS. Note that the difference between these two measures corresponds to the difference in CFL conditions between CTCS and the two finite-volume schemes. The linearized equations solved by FBL runs 60% faster than CTCS, and this is mainly due to the differences in their CFL conditions. In terms of memory requirement, all three non-linear schemes store the state vector $[\eta, hu, hv]$ for two successive

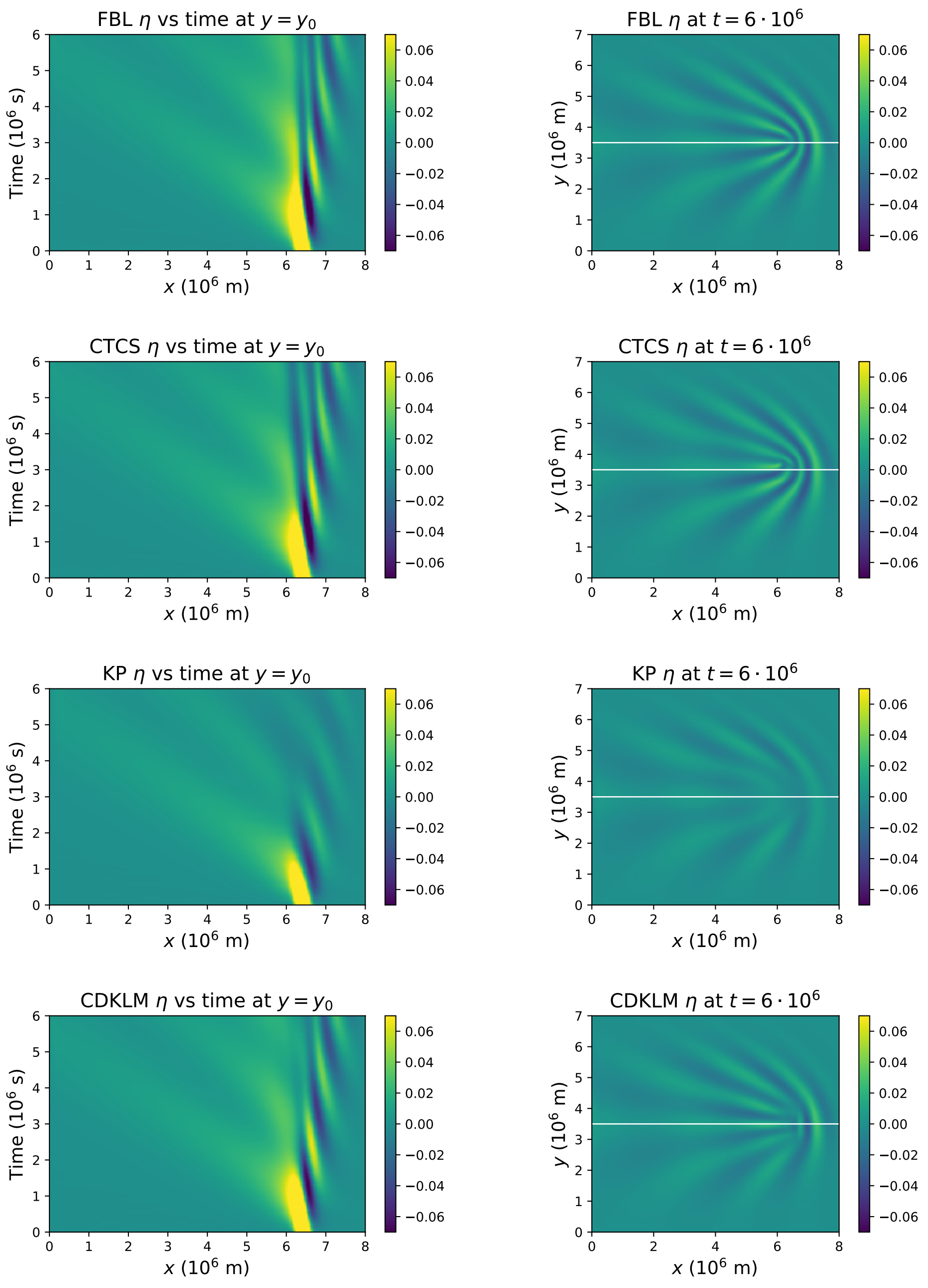
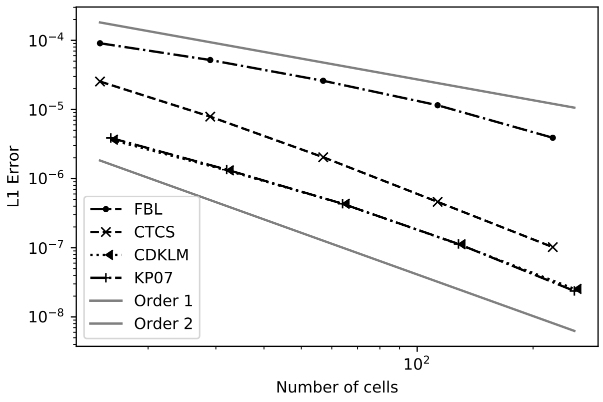
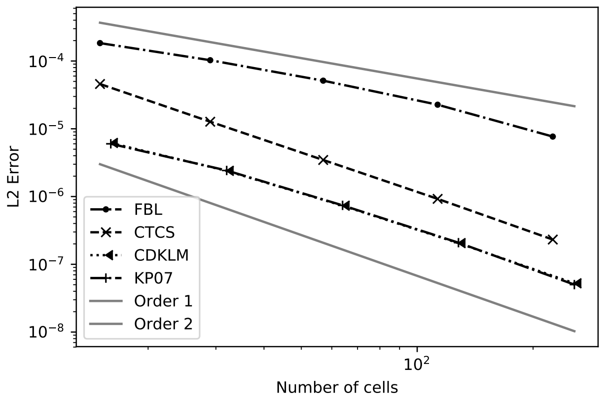


Figure 14: Case F: Long-term simulation results for topographic Rossby waves caused by the bathymetry given in (30). Hovmöller diagrams for y_0 are shown to the left, and η at the last time-step to the right. The white lines on the right-most figures illustrate the cross section at y_0. The solutions for FBL, CTCS, and CDKLM are very similar, whereas KP is more dissipative. The results here are more symmetric around y_0 than the planetary Rossby waves in Case E.





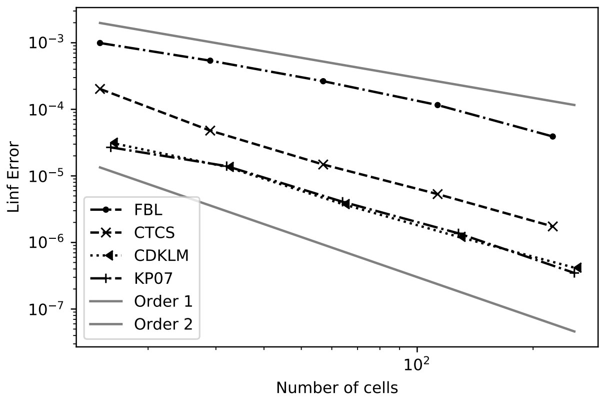


Figure 15: Numerical order of convergence for the four different schemes for η, displaying the expected L^1, L^2, and L^∞ error convergence rates (first order for FBL, and second order for the rest). The median of the L^1 convergence rate over the interval is 1.11, 2.09, 1.79, and 1.78 for FBL, CTCS, KP and CDKLM, respectively. The numerical L^2 convergence is 1.11, 1.94, 1.77, and 1.77, and for L^∞ the numerical convergence rate is 1.13, 1.70, 1.68, and 1.61.

Table 3: Comparison of numerical schemes, based on quantitative and qualitative properties. The rows are grouped as follows: General characteristics, oceanographic characteristics, and computational characteristics. The performance-related values are obtained from running a case consisting of 2048×2048 cells, and are normalized with respect to CTCS.

	Numerical Scheme			
	FBL	**CTCS**	**KP**	**CDKLM**
Preserve lake-at-rest	Yes	Yes	Yes	Yes
Accurately capture shocks	No	No	Yes	Yes
Case B and C: Rossby adjustment	Good	Good	Fairly good	Fairly good
Case D: Kelvin waves	Good	Medium	Medium	Good
Case E: Planetary Rossby waves	Good	Good	Medium	Good
Case F: Topographic Rossby waves	Good	Good	Medium	Good
Numerical L^2 convergence rate	1.11	1.94	1.77	1.77
Relative simulation wall time	0.4	1.0	4.3	5.7
Relative iterations per second	1.25	1.0	0.33	0.25
Memory req. for 2048×2048 cells	64.1 MB	112.3 MB	128.5 MB	128.5 MB

time-steps, whereas FBL only requires the state for one time-step. The finite-volume schemes are also implemented by storing the equilibrium depth H at both the cell centers and cell intersections, whereas the finite-volume schemes store H in the cell centers only. All these performance measures are summarized in Table 3.

4.8 Comparison

Table 3 summarizes the four schemes for the presented test cases. The table is divided into three groups, where the first group is related to general characteristics valid for rotational and non-rotational settings alike. The second group covers the oceanographic aspects, and the third reports on the numerical implementation.

All schemes conserve mass and momentum within the domain, and preserve lake-at-rest solutions. The finite-difference schemes, however, break down for solutions containing shocks. Even though shock solutions do not naturally occur in the ocean, the dam break test has been included here as it is highly useful to be aware of such limitations for the schemes at hand.

In the second group, we see that FBL and CTCS are able to capture the Rossby adjustment steady-state solution better than KP and CDKLM. All schemes except for KP are able to represent Kelvin waves of small amplitudes according to linear wave theory. Larger differences are seen as the amplitude is increased so that nonlinearities dominate the simulations. The finite-volume schemes capture the physical expected solution, with a shock forming in front of the wave. CTCS, however, gives a solution where the crest of the wave is slowed, resulting in a shock behind the wave. FBL maintains a linear solution, as is expected by the scheme. In the cases of planetary and topographic Rossby waves, FBL, CTCS and CDKLM give qualitatively the same solutions in accordance with theory, whereas KP is significantly more dissipative.

The final group considers computational aspects of the schemes. We note that the performance of CTCS is significantly better than KP and CDKLM, mainly due to more efficient scheme, and partly due to less strict CFL conditions. Finally, we note that all four schemes obtain the expected numerical order.

5 Summary and Conclusions

We have in this paper evaluated four numerical schemes for solving the rotational shallow-water equations by looking at six oceanographic test cases. The test cases have been chosen based on oceanographic concepts important for transient flow, and investigate each scheme's ability to produce and maintain geostrophic steady-state solutions (Rossby adjustment), capture fast coastal waves (Kelvin waves), and capture slow waves due to variations in the potential vorticity (Rossby waves). Two of the selected numerical schemes are the traditional finite-difference methods FBL and CTCS, evaluated on staggered grids arising from the oceanographic community. The last two schemes, KP and CDKLM, are recent finite-volume methods that have been developed within the community working on high-resolution schemes for more general hyperbolic conservation laws. These schemes have been chosen as they all have favorable properties with respect to doing ensemble-based simulations, e.g., for storm surges, and/or data assimilation for drift prediction.

Our findings show that the FBL scheme gives the expected linear results for all test cases, and is the most computational efficient scheme due to its simplicity. For the full nonlinear equations, we see that CTCS performs well in cases dominated by geostrophic balance, such as Rossby adjustment and Rossby waves, but gives incorrect dispersion for large gravity-driven waves. The naïve discretization of the Coriolis force in the KP scheme results in a too diffusive scheme, causing it to perform worse than the CDKLM scheme for all test cases. The CDKLM scheme, although more expensive than KP, gives good results on all test cases, and is shown to be superior for capturing gravity-driven motion, such as the Kelvin waves, but there are still potential for even better treatment of the geostrophic balances.

Code Availability

The software used in this research is published as supplementary material [47] under DOI 10.5281/zenodo.3204200.

Acknowledgements

This work is supported by the Research Council of Norway through grant number 250935 (GPU Ocean). The authors would also like to thank Knut-Andreas Lie and Lars Petter Røed for valuable comments and discussions.

A Oceanographic Background

The purpose of this appendix is to provide a physical outline of the planetary and topographic Rossby waves and the Rossby adjustment problem. All cases start with a “disturbance” in sea surface height and develop under strong dominance of the rotation. For the Rossby adjustment, one reaches a steady state, whereas the Rossby waves will create waves propagating out from the origin of the disturbance. Notably, the restoring force for rotationally dominated large waves is conservation of total potential vorticity. The following description can be found in several text books on large-scale fluid dynamics, but is provided here for the convenience of the readers.

A.1 Coriolis force

The Coriolis parameter is defined as

$$f = 2\Omega \sin(\phi), \tag{31}$$

in which Ω is the Earth’s rotation rate and ϕ is the latitude. Hence $|f|$ is zero at the equator and attains its maximum values on the poles. Denoting the horizontal velocity components (u, v) in the west-to-east (x) and south-to-north (y) directions, respectively, the Coriolis force per unit density is

$$\begin{aligned} F_f^{(x)} &= fv, \\ F_f^{(y)} &= -fu. \end{aligned} \tag{32}$$

The dominating force balance in the ocean is the geostrophic balance, which equates the pressure gradient force and the Coriolis force. For barotropic flows, the pressure gradient force is due to surface tilt. If we define the surface coordinate as $z = \eta(x, y, t)$, the barotropic geostrophic balance is then expressed mathematically as

$$\begin{aligned} -fv &= -g\frac{\partial \eta}{\partial x}, \\ fu &= -g\frac{\partial \eta}{\partial y}, \end{aligned} \tag{33}$$

in which g is the acceleration due to gravity.

A.2 Potential Vorticity

By neglecting friction and the nonlinear term in the pressure gradient, the equations for barotropic velocities $\boldsymbol{v}_h = (u, v)$ and continuity read

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} - fv = -g\frac{\partial \eta}{\partial x}, \tag{34}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + fu = -g\frac{\partial \eta}{\partial y}, \tag{35}$$

$$\frac{\partial \eta}{\partial t} + \frac{\partial}{\partial x}\left[(H + \eta)u\right] + \frac{\partial}{\partial y}\left[(H + \eta)v\right] = 0. \tag{36}$$

Taking the curl of (34) and (35), we obtain

$$\frac{d}{dt}(\zeta + f) = -(\zeta + f)\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right), \tag{37}$$

where $d/dt \equiv \partial/\partial t + u(\partial/\partial x) + v(\partial/\partial y)$ is the total derivative and $\zeta = \partial v/\partial x - \partial u/\partial y$ is the vertical component of the vorticity. (37) describes conservation of angular momentum. If depth does not change in time, we can rewrite (36) as

$$\frac{d}{dt}(H + \eta) = -(H + \eta)\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right). \tag{38}$$

Note the similarities between (37) and (38): both the vorticity and the total height of a material water column depend on the convergence/divergence of the flow field. Combining these equations, we find that

$$\frac{d}{dt}\left(\frac{\zeta + f}{H + \eta}\right) = 0. \tag{39}$$

The potential vorticity $(\zeta + f)/(H + \eta)$ is a conserved quantity in non-dissipative flows in a rotating reference frame. To clarify any discussion when using potential vorticity, ζ is generally referred to as the relative vorticity, and f as the planetary vorticity. (39) implies that the flow will tend to follow contours of constant f/H, since these are the dominating terms in geophysically relevant cases.

A.3 Kelvin waves

Waves influenced by earth rotation and "leaning" on a topography are known as Kelvin waves, and are very important for explaining a number of phenomena such as tides and coastal transports. Here we will accordingly consider a simple derivation of the main characteristics of these waves. Observations reveal that the velocity parallel to the coast is much larger than the velocities perpendicular to the coast. Assuming a coast along the x-axis and that $V \ll U$ (and assuming U, V have similar time scales) and that nonlinear terms are small we find the following governing equations from (34) and (35)

$$\frac{\partial U}{\partial t} = -gH\frac{\partial \eta}{\partial x}, \tag{40}$$

$$fU = -gH\frac{\partial \eta}{\partial y}, \tag{41}$$

$$\frac{\partial \eta}{\partial t} + \frac{\partial U}{\partial x} = 0. \tag{42}$$

Combining (40) and (42) gives

$$\frac{\partial^2 \eta}{\partial t^2} - gH\frac{\partial^2 \eta}{\partial x^2} = 0, \tag{43}$$

which shows that this system behaves as a wave solution moving with speed $c_0 = \pm\sqrt{gH}$ in the x-direction (later analysis will show that waves can only travel in positive x-direction). Combining (41) and (42) shows that the wave structure is governed by

$$\frac{\partial \eta}{\partial t} - \frac{gH}{f}\frac{\partial^2 \eta}{\partial x \partial y} = 0. \tag{44}$$

Here it may be noted that the parameter group in (43) can be written as the wave speed times the Rossby radius L_R,

$$L_R = \frac{\sqrt{gH}}{f} = \frac{c_0}{f}, \tag{45}$$

such that

$$\frac{\partial \eta}{\partial t} - L_R c_0 \frac{\partial^2 \eta}{\partial x \partial y} = 0. \tag{46}$$

The fact that the solution behaves as a wave in the x-direction, implies that the solution can be written as

$$\eta = G(y)F(x,t). \tag{47}$$

Thus, (46) gives

$$G\frac{\partial F}{\partial t} - L_R c_0 \frac{\partial G}{\partial y}\frac{\partial F}{\partial x} = 0, \tag{48}$$

or

$$\frac{\partial F/\partial t}{c_0 \partial F/\partial x} = L_R \frac{\partial G/\partial y}{G}. \tag{49}$$

The solutions are of the form

$$\eta \propto e^{y/L_R} F(x + c_0 t) + e^{-y/L_R} F(x - c_0 t). \tag{50}$$

Assuming that our solution must be limited as $y \to \infty$ implies that the solution with positive exponent must be zero. Thus, for problems unbounded in the y-direction

$$\begin{aligned} \eta &= e^{-y/L_R} F(x - c_0 t), \\ u &= \frac{g}{f L_R} F(x - c_0 t). \end{aligned} \tag{51}$$

These waves – referred to as Kelvin waves – travel in positive x-direction requires a coast on the right of their travel directions (for the northern hemisphere). Due to the rotation they are trapped within the Rossby radius from the coast.

A.4 Rossby Waves

Rotating systems support potential vorticity waves, or so-called Rossby waves. The dynamics of Rossby waves can more easily be studied when we assume $\eta = 0$, that is, using a rigid lid approximation. We may then introduce a stream function ψ such that

$$Hu = -\frac{\partial \psi}{\partial y}, \tag{52}$$

$$Hv = \frac{\partial \psi}{\partial x}, \tag{53}$$

which by definition fulfills the continuity equation

$$\frac{\partial}{\partial x}(Hu) + \frac{\partial}{\partial y}(Hv) = 0. \tag{54}$$

The conservation equation for potential vorticity, (39) with the rigid lid approximation, becomes

$$\frac{1}{H}\frac{\partial \zeta}{\partial t} + u\frac{\partial}{\partial x}\left(\frac{f}{H}\right) + v\frac{\partial}{\partial y}\left(\frac{f}{H}\right) = 0, \tag{55}$$

under the assumption $f \gg \zeta$. Using the definitions of the vorticity and stream function, the conservation of vorticity reads (see also [48], Eq. 10.12.4, p. 409)

$$\frac{\partial}{\partial t}\left[\frac{\partial}{\partial x}\left(\frac{1}{H}\frac{\partial \psi}{\partial x}\right) + \frac{\partial}{\partial y}\left(\frac{1}{H}\frac{\partial \psi}{\partial y}\right)\right] + \frac{f}{H^2}\left[\frac{\partial H}{\partial x}\frac{\partial \psi}{\partial y} - \frac{\partial H}{\partial y}\frac{\partial \psi}{\partial x}\right] + \frac{1}{H^2}\left[\frac{\partial f}{\partial x}\frac{\partial \psi}{\partial y} + \frac{\partial f}{\partial y}\frac{\partial \psi}{\partial x}\right] = 0. \tag{56}$$

Let us consider the case with $H = H(y)$ and $f = f_0 + \beta y$, where the latter is the so-called beta-plane approximation. Then $\partial f/\partial y = \beta$, and (56) becomes

$$\frac{\partial}{\partial t}\left[\frac{1}{H}\left(\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2}\right) - \frac{1}{H^2}\frac{\partial H}{\partial y}\frac{\partial \psi}{\partial y}\right] - \frac{f}{H}\left[\frac{1}{H}\frac{\partial H}{\partial y} + \frac{\beta}{f}\right]\frac{\partial \psi}{\partial x} = 0. \tag{57}$$

To investigate Rossby wave dispersion further, we assume that the depth is given by the function $H = H_0 \exp(-\alpha y)$. We also assume a wave solution $\psi = A \exp[i(kx + \kappa y - \omega t)]$, where A is a constant and $\kappa = l + i\gamma$ is a complex wave number in the y-direction. From (57) we obtain the dispersion relation

$$\kappa^2 - i\alpha\kappa + \left(\frac{k}{\omega}(\alpha f + \beta) + k^2\right) = 0. \tag{58}$$

From the complex part we obtain $\gamma = \alpha/2$. The real part yields the dispersion relation

$$\omega = -\frac{k(\alpha f + \beta)}{k^2 + l^2 + (\alpha/2)^2}. \tag{59}$$

With $\alpha \neq 0$ and $\beta = 0$, we have *topographic* Rossby waves, whereas with $\alpha = 0$ and $\beta \neq 0$, we have *planetary* Rossby waves.

From (58), we see there is a close similarity between these types of Rossby waves when $\alpha f = \beta$, in particular for small $\alpha^2 \ll k^2, l^2$. The zonal (x) component of planetary Rossby waves always propagate from east to west, while the topographic Rossby waves in the examples shown here propagate with shallow-water on their right. The group velocity and the energy propagation generally depends on the wave number, but can be in either direction.

A.5 Rossby Adjustment

The derivation of the equations describing the end state in the case of Rossby adjustment takes a different path. The linearized basic shallow-water equations read

$$\frac{\partial U}{\partial t} - fV = -gH\frac{\partial \eta}{\partial x}, \tag{60}$$

$$\frac{\partial V}{\partial t} + fU = -gH\frac{\partial \eta}{\partial y}, \tag{61}$$

$$\frac{\partial \eta}{\partial t} + \frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} = 0. \tag{62}$$

Here, $(U, V) = (hu, hv)$ is the volume transport. To simplify the derivation we assume that the depth is constant.

We aim to find a single equation for η, and take the divergence of (60) and (61), and the time derivative of (62). We can then eliminate U and V and obtain

$$\frac{\partial^2 \eta}{\partial t^2} + \nabla \cdot \left(c_0^2 \nabla \eta\right) + Hf\zeta = 0, \tag{63}$$

in which $c_0^2 = gH$. An equation for the vorticity can be obtained by taking the curl of (60) and (61), and combining with the continuity equation (62):

$$H\frac{\partial \zeta}{\partial t} - f\frac{\partial \eta}{\partial t} = 0. \tag{64}$$

Using (64) to eliminate ζ in (63), we obtain (see also [7], Eq. 3.6.9, p. 69)

$$\frac{\partial}{\partial t}\left[\left(\frac{\partial^2}{\partial t^2} + f^2\right) - \nabla \cdot \left(c_0^2 \nabla \eta\right)\right] = 0. \tag{65}$$

It is straightforward to integrate this equation in time, and assuming zero initial velocities and that $\eta = \eta_0$, we have

$$\frac{\partial^2 \eta}{\partial t^2} - c_0^2 \nabla^2 \eta + f^2(\eta - \eta_0) = 0. \tag{66}$$

The transient solution to this equation describes long surface waves modified by rotation, usually referred to as Poincaré waves. The steady solution $\bar{\eta}$ is the result of adjustment to the local rotation as a permanent modification of the mean surface level with velocities in geostrophic balance. This process is referred to as Rossby adjustment. The steady state is found by solving the Klein-Gordon equation that results from removing the time dependence in (66):

$$-c_0^2 \nabla^2 \bar{\eta} + f^2(\bar{\eta} - \eta_0) = 0. \tag{67}$$

The length scale given by $L_R = c_0/f$ is the Rossby radius of deformation, within which a disturbance will be trapped by rotation and the steady-state solution will be dominated by a geostrophic balance.

References

[1] F. Mesinger and A. Arakawa. *Numerical Methods Used in Atmospheric Models*, volume 1 of *GARP publications series*. World Meteorological Organization, International Council of Scientific Unions, 1976.

[2] A. Kurganov and G. Petrova. A second-order well-balanced positivity preserving central-upwind scheme for the Saint-Venant system. *Communications in Mathematical Sciences*, 5:133–160, 2007.

[3] A. Brodtkorb, M. Sætra, and M. Altinakar. Efficient shallow water simulations on GPUs: Implementation, visualization, verification, and validation. *Computers & Fluids*, 55(0):1–12, 2012.

[4] A. Chertock, M. Dudzinski, A. Kurganov, and M. Lukácová-Medvidová. Well-balanced schemes for the shallow water equations with Coriolis forces. *Numerische Mathematik*, 2017.

[5] E. Toro. *Shock-Capturing Methods for Free-Surface Shallow Flows*. John Wiley & Sons, Ltd., 2001.

[6] R. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2002.

[7] J. Pedlosky. *Geophysical Fluid Dynamics*. Springer New York, 1987.

[8] L. Røed. *Atmospheres and Oceans on Computers*. Springer International Publishing, 2018.

[9] J. Zhou. *Lattice Boltzmann Methods for Shallow Water Flows*. Springer-Verlag Berlin Heidelberg, 2004.

[10] B. Cockburn, G. Karniadakis, and C.-W. Shu. The development of discontinuous Galerkin methods. In B. Cockburn, G. Karniadakis, and C.-W. Shu, editors, *Discontinuous Galerkin Methods*, pages 3–50, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[11] R. Nair, S. Thomas, and R. Loft. A discontinuous Galerkin global shallow water model. *Monthly Weather Review*, 133(4):876–888, 2005.

[12] S. Noelle, N. Pankratz, G. Puppo, and J. Natvig. Well-balanced finite volume schemes of arbitrary order of accuracy for shallow water flows. *Journal of Computational Physics*, 213(2):474–499, 2006.

[13] W.-Y. Liang, T.-J. Hsieh, M. Satria, Y.-L. Chang, J.-P. Fang, C.-C. Chen, and C.-C. Han. A GPU-based simulation of tsunami propagation and inundation. In *Algorithms and Architectures for Parallel Processing*, volume 5574 of *Lecture Notes in Computer Science*, pages 593–603, Berlin/Heidelberg, 2009. Springer Verlag.

[14] M. Lastra, J. Mantas, C. Ureña, M. Castro, and J. García-Rodríguez. Simulation of shallow-water systems using graphics processing units. *Mathematics and Computers in Simulation*, 80(3):598–618, 2009.

[15] M. de la Asunción, J. Mantas, and M. Castro. Simulation of one-layer shallow water systems on multicore and CUDA architectures. *The Journal of Supercomputing*, 58(2):206–214, Nov 2011.

[16] A. Brodtkorb and M. Sætra. Explicit shallow water simulations on GPUs: Guidelines and best practices. In *XIX International Conference on Water Resources, CMWR 2012, June 17–22, 2012*. University of Illinois at Urbana-Champaign, 2012.

[17] K. Mandli, A. Ahmadia, M. Berger, D. Calhoun, D. George, Y. Hadjimichael, D. Ketcheson, G. Lemoine, and R. LeVeque. Clawpack: building an open source ecosystem for solving hyperbolic PDEs. *PeerJ Computer Science*, 2:e68, 2016.

[18] X. Qin, R. LeVeque, and M. Motley. Accelerating wave-propagation algorithms with adaptive mesh refinement using the graphics processing unit (GPU). *CoRR*, abs/1808.02638, 2018.

[19] C. Huxley and B. Syme. TUFLOW GPU-best practice advice for hydrologic and hydraulic model simulations. In *37th Hydrology & Water Resources Symposium 2016: Water, Infrastructure and the Environment*, pages 195–203. Engineers Australia, 2016.

[20] MIKE Powered by DHI. MIKE 21 graphical processing units (GPU) benchmarking report. Technical report, DHI, 2019. `https://www.mikepoweredbydhi.com/-/media/shared%20content/mike%20by%20dhi/flyers%20and%20pdf/product-documentation/gpu-benchmarking-report.pdf`, accessed: 2019-05-21.

[21] O. Delestre, C. Lucas, P.-A. Ksinant, F. Darboux, C. Laguerre, T.-N.-T. Vo, F. James, and S. Cordier. SWASHES: a compilation of shallow water analytic solutions for hydraulic and environmental studies. *International Journal for Numerical Methods in Fluids*, 72(3):269–300, 2013.

[22] W. Thacker. Some exact solutions to the nonlinear shallow-water wave equations. *Journal of Fluid Mechanics*, 107:499–508, 1981.

[23] S. Frazão, X. Sillen, and Y. Zech. Dam-break flow through sharp bends, physical model and 2D Boltzmann model validation. In *The Proceedings of the 1st CADAM meeting*, 1998.

[24] S. Frazão, F. Alcrudo, and N. Goutal. Dam-break test cases summary. In *The Proceedings of the 4th CADAM meeting*, 1999.

[25] N. Goutal and F. Maurel. Proceedings of the 2nd Workshop on Dam-Break Wave Simulation. Technical report, Groupe Hydraulique Fluviale, Département Laboratoire National d'Hydraulique, Electricité de France, 1997.

[26] N. Goutal. The Malpasset dam failure, an overview and test case definition. In *The Proceedings of the 4th CADAM meeting*, 1999.

[27] J.-M. Hervouet and A. Petitjean. Malpasset dam-break revisited with two-dimensional computations. *Journal of Hydraulic Research*, 37:777–788, 1999.

[28] A. Valiani, V. Caleffi, and A. Zanni. Case study: Malpasset dam-break simulation using a two-dimensional finite volume method. *Journal of Hydraulic Engineering*, 128:460–472, 2002.

[29] D. Williamson, J. Drake, J. Hack, R. Jakob, and P. Swarztrauber. A standard test set for numerical approximations to the shallow water equations in spherical geometry. *Journal of Computational Physics*, 102(1):211–224, 1992.

[30] J. Galewsky, R. Scott, and L. Polvani. An initial-value problem for testing numerical models of the global shallow-water equations. *Tellus A: Dynamic Meteorology and Oceanography*, 56(5):429–440, 2004.

[31] R. Comblen, J. Lambrechts, J.-F. Remacle, and V. Legat. Practical evaluation of five partly discontinuous finite element pairs for the non-conservative shallow water equations. *International Journal for Numerical Methods in Fluids*, 63(6):701–724, 2010.

[32] G. Tumolo, L. Bonaventura, and M. Restelli. A semi-implicit, semi-Lagrangian, p-adaptive discontinuous Galerkin method for the shallow water equations. *Journal of Computational Physics*, 232(1):46–67, 2013.

[33] A. Shchepetkin and J. McWilliams. The regional oceanic modeling system (ROMS): A split-explicit, free-surface, topography-following-coordinate oceanic model. *Ocean Modelling*, 9(4):347–404, 2005.

[34] R. Asselin. Frequency filter for time integrations. *Monthly Weather Review*, 100(6):487–490, 1972.

[35] A. Sielecki. An energy-conserving difference scheme for the storm surge equations. *Monthly Weather Review*, 96(3):150–156, 1968.

[36] L. Røed. Documentation of simple ocean models for use in ensemble predictions. Part I: Theory. Technical report, Norwegian Meteorological Institute, 2012.

[37] N. Phillips. An example of non-linear computational instability. *The Atmosphere and the Sea in motion*, 501, 1959.

[38] A. Robert, F. Shuman, and J. Gerrity. On partial difference equations in mathematical physics. *Monthly Weather Review*, 98(1):1–6, 1970.

[39] S. Gottlieb, C.-W. Shu, and E. Tadmor. Strong stability-preserving high-order time discretization methods. *SIAM Review*, 43(1):89–112, 2001.

[40] P. Marchesiello, J. McWilliams, and A. Shchepetkin. Open boundary conditions for long-term integration of regional oceanic models. *Ocean Modelling*, 3(1):1–20, 2001.

[41] H. Davies. A lateral boundary formulation for multi-level prediction models. *Quarterly Journal of the Royal Meteorological Society*, 102(432):405–418, 1976.

[42] E. Martinsen and H. Engedahl. Implementation and testing of a lateral boundary scheme as an open boundary condition in a barotropic ocean model. *Coastal Engineering*, 11(5):603–627, 1987.

[43] W. Blumen. Geostrophic adjustment. *Reviews of Geophysics*, 10(2):485–528, 1972.

[44] L. Mendez-Nunez and J. Carroll. Comparison of leapfrog, Smolarkiewicz, and MacCormack schemes applied to nonlinear equations. *Monthly Weather Review*, 121(2):565–578, 1993.

[45] R. Grotjahn and J. O'Brien. Some inaccuracies in finite differencing hyperbolic equations. *Monthly Weather Review*, 104(2):180–194, 1976.

[46] H. Holm, A. Brodtkorb, and M. Sætra. Performance and energy efficiency of CUDA and OpenCL for GPU computing using Python. *Advances in Parallel Computing*, 36:593–604, 2020.

[47] H. Holm, A. Brodtkorb, M. Sætra, G. Broström, and K. Christensen. Supplementary material to evaluation of selected finite-difference and finite-volume approaches to rotational shallow-water flow. DOI: 10.5281/zenodo.3204200, 5 2019.

[48] A. Gill. *Atmosphere-Ocean Dynamics*. International Geophysics. Academic Press, 1982.

Paper II

GPU Computing with Python: Performance, Energy Efficiency and Usability

Håvard Heitlo Holm, André Rigland Brodtkorb, Martin Lilleeng Sætra

Computation, volume 8, number 1:4, 2020
(Special issue on Energy-Efficient Computing on Parallel Architectures)
DOI: 10.3390/computation8010004

GPU Computing with Python: Performance, Energy Efficiency and Usability

Håvard H. Holm*[1,2], André R. Brodtkorb[3,4], and Martin L. Sætra[3,4]

[1] SINTEF Digital, Mathematics and Cybernetics, P.O. Box 124 Blindern, NO-0314 Oslo, Norway.
[2] Norwegian University of Science and Technology, Department of Mathematical Sciences, NO-7491 Trondheim, Norway.
[3] Norwegian Meteorological Institute, P.O. Box 43 Blindern, NO-0313 Oslo, Norway.
[4] Oslo Metropolitan University, Department of Computer Science, P.O. Box 4 St. Olavs plass, NO-0130 Oslo, Norway.

Abstract

In this work, we examine the performance, energy efficiency and usability when using Python for developing HPC codes running on the GPU. We investigate the portability of performance and energy efficiency between CUDA and OpenCL; between GPU generations; and between low-end, mid-range and high-end GPUs. Our findings show that the impact of using Python is negligible for our applications, and furthermore, CUDA and OpenCL applications tuned to an equivalent level can in many cases obtain the same computational performance. Our experiments show that performance in general varies more between different GPUs than between using CUDA and OpenCL. We also show that tuning for performance is a good way of tuning for energy efficiency, but that specific tuning is needed to obtain optimal energy efficiency.

1 Introduction

GPU computing was introduced in the early 2000s, and has since become a popular concept. The first examples were acceleration of simple algorithms such as matrix-matrix multiplication by rephrasing the algorithm as operations on graphical primitives (see e.g., [2]). This was cumbersome and there existed no development tools for general-purpose computing. However, many algorithms were implemented on the GPU as proof-of-concepts, showing large speedups over the CPU [3]. Today, the development environment for GPU computing has evolved tremendously and is both mature and stable: Advanced debuggers and profilers are available, making debugging, profile-driven development, and performance optimization easier than ever.

The GPU has traditionally been accessed using compiled languages such as C/C++ or Fortran for the CPU code, and a specialized programming language for the GPU. The rationale is often that performance is paramount, and that compiled languages are therefore required. However, for many GPU codes, most of the time is spent in the numerical code running on the GPU. In these cases, we can possibly use a higher-level language such as Python for the program flow without significantly affecting the performance. The benefit is that using higher-level languages might increase productivity [4]. PyCUDA and PyOpenCL [5] are two Python packages that offer access to CUDA and OpenCL from Python, and have become mature and popular packages since their initial release nearly ten years ago.

Herein, we compare the performance, energy efficiency, and usability of PyCUDA and PyOpenCL for important algorithmic primitives found in many HPC applications [6]: a memory bound numerical simulation code and a computationally bound benchmark code. The memory bound code performs simulation of the shallow-water equations using explicit stencils [7, 8], and represents a class of problems that are particularly well suited for GPU computing [9, 10, 11, 12, 13]. The computationally bound code computes the Mandelbrot set, and we use it to explore limitations of using Python for GPU computing.

*Corresponding author: havard.heitlo.holm@sintef.no
This paper is an extended version of our paper published in International Conference on Parallel Computing (ParCo2019) [1]

We show that accessing the GPU from Python is as efficient as from C/C++ in many cases, demonstrate how profile-driven development in Python is essential for increasing performance for GPU code (up to 5 times), and show that the energy efficiency increases proportionally with performance tuning. Finally, we investigate the portability of the improvements and power efficiency both between CUDA and OpenCL and between different GPUs. Our findings are summarized in tables that justify that using Python can be preferable to C++, and that using CUDA can be preferable to using OpenCL. Our observations should be directly transferable to other similar architectures and problems.

2 Related Work

There are several high-level programming languages and libraries that offer access to the GPU for certain sets of problems and algorithms. OpenACC [14] is one example which is pragma-based and offers a set of directives to execute code in parallel on the GPU. However, such high-level abstractions are typically only efficient for certain classes of problems and are often unsuitable for non-trivial parallelization or data movement. CUDA [15] and OpenCL [16] are two programming languages that offer full access to the GPU hardware, including the whole memory subsystem. This is an especially important point, since memory movement is a key bottleneck in many numerical algorithms [6] and therefore has a significant impact on energy consumption.

The performance of GPUs has been reported extensively [17], and several authors have shown that GPUs are efficient in terms of energy-to-solution. Huang et al. [18] demonstrated early on that GPUs could not only speed up computational performance, but also increase power efficiency dramatically using CUDA. Qi et al. [19] show how OpenCL on a mobile GPU can increase performance of the discrete Fourier transform by 1.4 times and decrease the energy use by 37%. Dong et al. [20] analyze the energy efficiency of GPU BLAST which simulates compressible hydrodynamics using finite elements with CUDA and report a 2.5 times speedup and a 42% increase in energy efficiency. Klôh [21] report that there is a wide spread in terms of energy efficiency and performance when comparing 3D wave propagation and full waveform inversion on two different architectures. They compare an Intel Xeon coupled with an ARM-based Nvidia Jetson TX2 GPU module, and find that the Xeon platform performs best in terms of computational speed, whilst the Jetson platform is most energy efficient. Memeti et al. [22] compare the programming productivity, performance, and energy use of CUDA, OpenACC, OpenCL and OpenMP for programming a system consisting of a CPU and GPU or a CPU and an Intel Xeon Phi coprocessor. They report that CUDA, OpenCL and OpenMP have similar performance and energy consumption in one benchmark, and that OpenCL performs better than OpenACC for another benchmark. In terms of productivity, the actual person writing the code is important, but OpenACC and OpenMP require less effort than CUDA and OpenCL, and CUDA can require significantly less effort than OpenCL.

Previous studies have also shown that CUDA and OpenCL can compete in terms of performance as long as the comparison is fair [23, 24, 25, 26], and there have also been proposed automatic source to source compilers that report similar results [27, 28]. We do not know of any other established literature that thoroughly compares the performance, usability and energy efficiency of CUDA and OpenCL when accessed from Python.

3 GPU Computing in Python

In this work, we focus on using Python to access the GPU through CUDA and OpenCL. These two GPU programming models are conceptually very similar, and both offer the same kind of parallelism primitives. The main idea is that the computational domain is partitioned into equally sized subdomains that are executed independently and in parallel. Even though the programming models are similar, their terminology differs slightly, and in this paper we will use that of CUDA. A full review is outside the scope of this work, but can be found in [29, 30]. The following sections give an overview of important parts of CUDA and OpenCL, and discuss their respective Python wrappers. We give a short introduction to using them from C++ and Python, and compare the benefits and drawbacks of both approaches.

3.1 CUDA

CUDA [15] (Compute Unified Device Architecture) was first released in 2007, and is available on all Nvidia GPUs as Nvidia's proprietary GPU computing platform. It includes third-party libraries and integrations, the directive-based

OpenACC [14] compiler, and the CUDA C/C++ programming language. Today, five of the ten fastest supercomputers (including number one) use Nvidia GPUs, as well as nine out of the ten most energy-efficient [31].

CUDA is implemented in the Nvidia device driver, but the compiler (`nvcc`) and libraries are packaged in the CUDA toolkit and SDK.[1] The toolkit and SDK contain a plethora of examples and libraries. In addition, the toolkit contains Nvidia Nsight, which is an extension for Microsoft Visual Studio (Windows) and Eclipse (Linux) for interactive GPU debugging and profiling. Nsight offers code highlighting, unified CPU and GPU trace of the application, and automatic identification of GPU bottlenecks. The Nvidia Visual Profiler is a stand-alone cross-platform application for profiling of CUDA programs, and CUDA versions for debugging (cuda-gdb) and memory checking (cuda-memcheck) also exist.

3.2 OpenCL

OpenCL [16] (Open Compute Language) is a free and open heterogeneous computing platform that was initiated by Apple in 2009, and today the OpenCL standard is maintained and developed by the Khronos group. Whilst CUDA is made specifically for Nvidia GPUs, OpenCL can run on a number of heterogeneous computing architectures including GPUs, CPUs, FPGAs, and DSPs. The OpenCL API is defined in a common C/C++ header, and a runtime library (ICD loader) redirects OpenCL calls to the appropriate device driver by using an installable client driver (ICD), specific to each architecture. The actual OpenCL is therefore implemented in each vendor's device driver. Contrary to CUDA, there is no common toolkit, but there are several third-party libraries.[2]

Profiling an OpenCL application can be challenging, and the available tools vary depending on your operating system and hardware vendor.[3] It is possible to get timing information on kernel and memory transfer operations by enabling event profiling information and adding counters explicitly in your source code. This requires extra work and makes the code more complex. Visual Studio can measure the amount of run time spent on the GPU, and CodeXL [32] can be used to get more information on AMD GPUs. CodeXL is a successor to gDebugger which offers features similar to those found in Nsight in addition to power profiling, and is available both as a stand-alone cross-platform application and as a Visual Studio extension. While it is possible to use Visual Profiler for OpenCL, this requires the use of the command-line profiling functionality in the Nvidia driver, which needs to be enabled through environment variables and a configuration file. After running the program with the profiling functionality in effect, the profiling data can be imported into Visual Profiler. Intel Code Builder (part of Intel SDK for OpenCL Applications [33]) and Intel Vtune Amplifier [34] can also be used for OpenCL debugging and profiling, but these tools only support Intel CPUs and Intel Xeon Phi processors.

One disadvantage of OpenCL is that there are large differences between the OpenCL implementations from different vendors, and good performance is likely to rely on vendor-specific extensions. One example is that OpenCL 2.2 is required for using C++ templates in the GPU code, but vendors such as Nvidia only support OpenCL version 1.2. It should also be mentioned that OpenCL has been deprecated in favour of Metal [35] by Apple in their most recent versions of Mac OS X.

3.3 GPU Computing from Python

Researchers spend a large portion of their time writing computer programs [36], and compiled languages such as C/C++ and Fortran have been the de facto standard within scientific computing for decades. These languages are well established, well documented, and give access to a plethora of native and third-party libraries. C++ is the standard way of accessing CUDA and OpenCL today, but developing code with these languages is time consuming and requires great care. Using higher-level languages such as Python can significantly increase development productivity [4, 37]. However, it should be mentioned that the OpenCL and CUDA kernels themselves are not necessarily made neither simpler nor shorter by using Python: The productivity gain comes instead from Python's less verbose code style for the CPU part of the code. This influences every part of the host code, from boilerplate initialization code and data pre-processing, to CUDA/OpenCL API calls, post-processing, and visualization of results.

Since Python was first released in 1994, it has gained momentum within scientific computing. Python is easy to learn, powerful, and emphasizes readability of code. All together, this allows for efficient prototyping of code, but

[1]Available at `https://developer.nvidia.com/cuda-zone`

[2]For a list of OpenCL resources, see `https://www.khronos.org/opencl/resources`

[3]An extensive list of OpenCL debugging and profiling tools can be found at `https://www.khronos.org/opengl/wiki/Debugging_Tools`

unfortunately often at the expense of performance. Since Python is an interpreted language, it does not have the speed of C/C++ and Fortran. For example, a for loop in C/C++ or Fortran will execute at great speed, whilst this construction is relatively slow in Python. This is acceptable for many application areas, but for computationally intensive codes it becomes a showstopper.

A popular approach to combining the speed of compiled code with Python is to have the program flow written in Python, and the performance critical inner loop in a compiled language. The performance critical code can then be called from Python, using e.g., SWIG wrappers, Numba [38] or Cython [39]. The following example illustrates this by computing $C = A'A$ with DGEMM from the BLAS library supplied with SciPy:

```
import numpy as np
import scipy.linalg.blas as blas
N = 2048
A = np.ones((N,N))
C = blas.dgemm(alpha=1.0, a=A.T, b=A)
```

Even though Python can be relatively slow, most of the time in this example is spent in the very efficient BLAS library, giving good overall performance. This approach thus offers the speed of compiled code together with the productivity of a high-level language. We can in a similar way execute GPU code from Python, and our experiments show that this gives good overall performance. Hence, Python can also be a viable option for production-level code, not only for experimental code and prototypes.

There are several libraries that today offer access to the GPU from Python. One class of libraries are such as OpenCV [40] that offers GPU acceleration of algorithms within a specific field. Such libraries are outside the scope of this work, as we focus on general-purpose GPU computing. In addition to these types of libraries, Numba [38] and CuPy [41] are general-purpose programming environments in Python that offer full access to the GPU. However, the GPU programs in Numba are written as Python functions, and the programmer has to rely on Numba for efficient parallelization of the code. While such a design lowers the bar for developers to write code that executes on the GPU, details that are crucial for obtaining the full potential performance might be lost in the abstraction. Additionally, Numba is missing support for dynamic parallelism and texture memory. CuPy also offers functionality to define the GPU functions in terms of Python code, but additionally supports raw kernels written in native CUDA.

PyCUDA and PyOpenCL [5] are Python packages that offer access to CUDA and OpenCL, respectively. Both libraries expose the complete API of the underlying programming models, and aim to minimize the performance impact. The GPU kernels, which are crucial for the inner loop performance, are written in native low-level CUDA or OpenCL, and memory transfers and kernel launches are made explicit through Python functions. The result is an environment suitable for rapid prototyping of high-performing GPU code. Listing 1 shows a minimal example for filling an array with the numbers 1 to N. The first four lines import PyCUDA and numpy. Lines six to ten hold the CUDA source code (written in CUDA C/C++), and compiles it using the `SourceModule` interface. Line eleven retrieves a handle to the kernel so that we can call the GPU function from Python. Line 14 allocates data in Python using numpy, which is then handed over to the GPU kernel in line 15. This manages automatic uploading and downloading of data from the GPU through the `drv.Out` class. The example shows how PyCUDA can be used to run a GPU kernel in less than 20 lines of code. The equivalent C++ code would be far longer.

PyCUDA and PyOpenCL can be installed using popular package managers such as apt-get, pip, and conda. However, the fact that they also require that CUDA and OpenCL are properly installed makes things a bit more complicated. The installation procedure on Windows is especially tricky, and it can be a challenge to install the packages successfully. On Linux, a challenge with OpenCL is that Nvidia only supports version 1.2, whilst PyOpenCL tries to compile with version 2.0 by default when it is installed with pip. This results in runtime crashes when loading the PyOpenCL package. The solution is to manually compile the package, which can be cumbersome. An alternative is to use the packages in apt-get, but these are often very outdated. For PyCUDA, however, we have experienced less difficulties installing on Linux.

PyOpenCL ships with integration for Jupyter Notebooks [42], which enables rapid prototyping in an interactive REPL environment[4]. Using the Jupyter Notebook for prototyping is extremely efficient, and our development cycle has typically been as follows:

[4]REPL stands for read-eval-print loop, and is a class of interactive development environments where you execute the code as you write it.

Listing 1: Programming example that shows how to fill an array with the numbers 1 to N using PyCUDA. The corresponding C++ code is much longer.

```
import pycuda.autoinit
import pycuda.driver as drv
import pycuda.compiler as compiler
import numpy as np

module = compiler.SourceModule("""
__global__ void fill(float *dest) {
    int i = blockIdx.x*blockDim.x+threadIdx.x;
    dest[i] = i+1;
} """)
fill = module.get_function("fill")

N = 200
cpu_data = np.empty(N, dtype=np.float32)
fill(drv.Out(cpu_data), block=(N,1,1), grid=(1,1,1))
print(cpu_data)
```

1. Prototype and develop code in a Jupyter Notebook.
2. Clean up code in notebook.
3. Move code from notebook to separate Python modules.

The first stage often entails developing the idea and concept interactively, making test programs, and plotting intermediate and final results. Then, when we have a first implementation, we continue by cleaning up the code and making it suitable for moving into a separate Python module. The tests are added to an appropriate continuous integration environment.

Unfortunately, we have experienced that OpenCL occasionally crashes at random within the Jupyter Notebook environment. After a significant amount of debugging, we discovered that the context was not properly cleaned up, and variables that went out of scope were not properly deallocated, thereby causing crashes for longer Jupyter sessions. Unfortunately, these crashes demanded a reload of the GPU driver, which typically means rebooting the system. Our solution was simply to add explicit invocation of Python's garbage collector after dereferencing the variables pointing to GPU main memory.

After having worked with CUDA in the Jupyter Notebook, we discovered that similar problems were also present here. However, because CUDA is somewhat stateful, it uses a stack of contexts and this stack was not properly cleaned up. Our solution was in this case to write a context manager that performed the correct pushing and popping of the CUDA context stack. After having addressed these shortcomings of PyCUDA and PyOpenCL, we have not experienced any crashes caused by the interplay with the Jupyter Notebook.

3.4 C++ Versus Python

The GPU can be accessed both from C++ and Python, and these two approaches have their own benefits and drawbacks. In this section we will examine how a computationally bound application – computing the Mandelbrot set – can be implemented in the four combinations: C++ and CUDA; C++ and OpenCL; Python and CUDA; and Python and OpenCL.

The Mandelbrot set M consists of all complex points c in which

$$z_{n+1} = z_n^2 + c$$

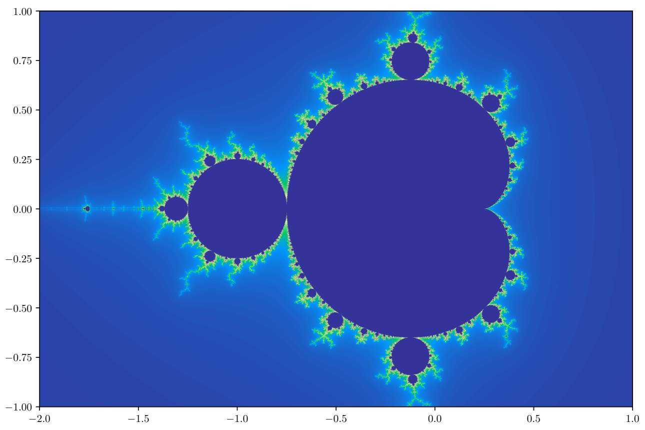


Figure 1: Mandelbrot set colored using continuous coloring. Each complex coordinate c is determined as within the set if the expression $z_{n+1} = z_n^2 + c$ does not diverge after a given set of iterations. The number of iterations performed for each thread is shown here using a continuous color scale. Locations close to the boundary perform many iterations, whilst only a few iterations are required far away. It is evident that threads in close proximity may perform very different number of iterations.

remains bounded as $n \to \infty$. In practice, one typically initializes a complex number c and $z_0 = 0$, and iterates until $|z_{n+1}| \geq 2$ or an upper limit of iterations has been reached:

```
//Loop until iterations or until it diverges
while (|z| < 2.0 && n < iterations) {
    z = z*z + c;
    ++n;
}
```

Our application consists of a GPU kernel which first initializes the complex coordinate, c, based on the position of each thread in the computational grid. Each thread then executes a loop as shown above before the output value for the thread is written to main memory. If we allow e.g., 1000 iterations, a single thread may perform up-to 1000 iterations of the above while-loop before it writes a single number to main GPU memory. However, as Figure 1 shows, the number of actual iterations performed by each thread varies dramatically, even between threads located close to each other. Between CUDA and OpenCL, the kernel code is close to identical, with only syntactic differences (see also Table 3). The actual kernel that runs on the GPU is the same for both the C++ and the Python variants, and difference between them are therefore found in the host code only.

The host code is responsible for allocating output data on the GPU, launching the kernel, and downloading the result from the GPU to the CPU. To benchmark the different variants, we compute the Mandelbrot set for different extents and measure how much time the CPU and GPU uses for different sections of the code. It should be noted that the overhead of running a traditional Python for-loop is significant, but can be reduced using e.g., so-called list comprehensions.

Table 1 shows a summary of our benchmarks. It shows clearly that the kernel launch overhead is larger for Python than for C++, but it becomes negligible when compared to the kernel execution time. Thus, for kernels that last more than a few ms there will be little performance benefit of using C++, and Python will typically be equally fast. The reason for this is that the CPU in both cases simply will be waiting for the GPU to complete execution for most of the time and the performance difference will be completely masked. This is in particular shown here in the “Wall time”

Table 1: Performance of CUDA and OpenCL for the Mandelbrot application from both Python and C++. The code was run on an Nvidia Tesla M2090 GPU to compute fifty consecutive zooms onto the point $-0.75 + 0.1i$ with a maximum of 5000 iterations. OpenCL 2.0 is available in the host section of the code, but the device section must still use OpenCL 1.2 for the C++ version. The development time is set subjectively by the authors and the lines of code metric contains only the CPU code related to the actual GPU kernel launch. On Windows we were unable to get asynchronous execution with OpenCL on several different machines, Python versions and GPU driver versions. We have not been able to pinpoint the cause of this, and suspect that asynchronous execution of OpenCL in Windows is not fully supported. We have used page locked memory with CUDA, whilst this was not easily available through OpenCL for the download. The overhead for OpenCL therefore includes the transfer as it cannot be run concurrently with other operations.[5] The wall time includes all the time the CPU spends from launching the first kernel to having completed downloading the last result from the GPU.

	C++		**Python**	
	CUDA	**OpenCL**	**CUDA**	**OpenCL**
API version	10.0	1.2 / 2.0	10.0	1.2
Development time	Medium	Medium	Fast	Fast
Approximate lines of code	145	130	100	100
Compilation time	~ 5 s	~ 5 s	~ 5 s	Interactive
Kernel launch overhead	13 µs	318 µs	19 µs	377 µs
Download overhead	9 µs	4007 µs	52 µs	8872 µs
Kernel GPU time	480 ms	446 ms	478 ms	444 ms
Download GPU time	4.0 ms	3.9 ms	4.0 ms	8.8 ms
Wall time	24.2 s	22.5 s	24.1 s	22.7 s

row, which shows no practical difference between C++ and Python. In fact, for CUDA the Python variant executes marginally faster. Our explanation to this is that PyCUDA automatically sets the compilation flags for `nvcc` for the specific GPU, yielding more optimized code.

One interesting difference between CUDA and OpenCL from Python is that the compilation time differs. PyCUDA uses `nvcc` to compile and link an executable in a similar fashion to a regular C++ program. PyOpenCL on the other hand, simply hands over the OpenCL kernel source code to the OpenCL driver which compiles it on-the-fly. We expect that incorporation of the recent Nvidia runtime compilation library (NVRTC) into PyCUDA will alleviate this shortcoming.

If we compare the other metrics, we see that both the development time and number of lines of code is significantly better for Python. Of particular note is that debugging and visualizing results becomes interactive when using Python and Jupyter Notebooks, thereby increasing the development efficiency dramatically. This is done using standard tools such as Matplotlib [43], making it extremely easy to visualize and explore results. In C++, on the other hand, the only way to reasonably explore the results is through file output (e.g., CSV) and plotting using third-party tools.

4 Porting, Profiling, and Benchmarking Performance and Energy Efficiency

We now describe the process of porting code between PyOpenCL and PyCUDA, and optimizating the PyCUDA versions through profile-driven development. We consider simulation of the shallow-water equations using three different numerical schemes:

- a linear finite difference scheme,
- a nonlinear finite difference scheme, and

[5] It should be noted that OpenCL should support efficient pinned memory transfers, but we were unable to get this to work using several driver versions on several different machines. We therefore report the observed transfer times without pinned memory.

Table 2: A list of the GPUs used in this work. The profile-driven development was carried out on the 840M and GTX780, and the reminding high-end GPUs were used for performance benchmarking. We have used the 840M, GTX780, P100 and V100 for benchmarking power efficiency. Note that the performance in gigaFLOPS is for single precision. The K80 GPU consists of two processors on the same card and has a boost feature for temporarily increasing the clock speed to increase performance to 2×4368 gigaFLOPS.

Model	**Class**	**Architecture (year)**	**Memory**	**GigaFLOPS**	**Bandwidth**	**Power device**
Tesla M2090	Server	Fermi (2011)	6 GiB	1331	178 GB/s	N.A.
Tesla K20	Server	Kepler (2012)	6 GiB	3524	208 GB/s	N.A.
GeForce GTX780	Desktop	Kepler (2013)	3 GiB	3977	288 GB/s	Watt meter
Tesla K80	Server	Kepler (2014)	2×6 GiB	2×2795	2×240 GB/s	N.A.
GeForce 840M	Laptop	Maxwell (2014)	4 GiB	790	16 GB/s	Watt meter
Tesla P100	Server	Pascal (2016)	12 GiB	9523	549 GB/s	nvidia-smi
Tesla V100	Server	Volta (2017)	16 GiB	14899	900 GB/s	nvidia-smi

- a high-resolution finite volume scheme.

The schemes are used for simulating real-world ocean currents, and two of them have been used operationally in the early days of computational oceanography. All three schemes are essentially stencil operations with an increasing level of complexity, and their details are summarized in Holm et al. [8].

The numerical schemes are algorithmically well suited for the GPU, but little effort has been made to thoroughly optimize the codes performance on a specific GPU. It is well known in the GPU computing community that performance is not portable between GPUs, neither for OpenCL nor CUDA, and automatically generating good kernel configurations is an active research area (see e.g., [44, 45, 46]). We start by porting the three schemes to CUDA, before using the available profiling tools for CUDA to analyze and optimize each scheme. The obtained optimizations are then also back-ported into the original OpenCL code. The profiling and tuning is carried out on a laptop with a dedicated GeForce 840M GPU, representing the low-end part of the GPU performance scale, and on a desktop with a mid-range GeForce GTX 780 GPU representing a typical mid-range GPU. We compare the performance of the original and optimized implementations with PyCUDA and PyOpenCL using seven GPUs listed in Table 2, which also includes several high-end server GPUs.

4.1 Porting from PyOpenCL to PyCUDA

The porting process requires changing both the kernel code that runs on the GPU, and the API calls in the CPU code. The kernels will in most cases run and produce correct results after a simple change of keywords. The CPU API calls, however, are quite different between CUDA and OpenCL, and require more attention. This includes handling devices, contexts and streams, compiling and linking kernels, setting kernel arguments, execution of kernels, and memory transfers between the CPU and GPU (or between GPUs).

A difficulty in the porting process is that it involves a significant amount of changes that all must be completed before it is possible to successfully compile, run and test the code. The Python interpreter and the CUDA compiler can be helpful in the porting process, as they will indicate the locations where code needs to be altered. A summary of key

Table 3: Keywords, functions and API calls in CUDA and OpenCL. In many cases a simple search and replace is sufficient to translate a program.

CUDA	**OpenCL**
Function qualifiers	
`__global__`	`__kernel`
`__device__`	`N/A`
Variable qualifiers	
`__constant__`	`__constant`
`__device__`	`__global`
`__shared__`	`__local`
Indexing	
`gridDim`	`get_num_groups()`
`blockDim`	`get_local_size()`
`blockIdx`	`get_group_id()`
`threadIdx`	`get_local_id()`
`blockIdx*blockDim+threadIdx`	`get_global_id()`
`gridDim*blockDim`	`get_global_size()`
Synchronization	
`__syncthreads()`	`barrier()`
`__threadfence()`	`N/A`
`__threadfence_block()`	`mem_fence()`
API calls	
`kernel<<<...>>>()`	`clEnqueueNDRangeKernel()`
`cudaGetDeviceProperties()`	`clGetDeviceInfo()`

differences between CUDA and OpenCL, which directly correspond to the steps below, can be found in Table 3. The steps needed to port our simulator from PyOpenCL to PyCUDA should also be applicable for porting other codes:

1. Import PyCUDA instead of PyOpenCL.
2. Change API calls from PyOpenCL to PyCUDA, paying extra attention to context and stream synchronization.
3. Adjust kernel launch parameters. Block sizes for PyCUDA need to be 3D and global sizes are given in number of blocks instead of total number of threads.
4. Use CUDA indexing in the kernels. Note that `gridDim` needs to be multiplied with `blockDim` to get the CUDA-equivalent of OpenCL `get_global_size()`.
5. Search and replace the remaining keywords in the kernels. Note that GPU functions in OpenCL have no special qualifier and that GPU main memory pointers need no qualifier for function arguments in CUDA.

Even though it might be straightforward to port codes between CUDA and OpenCL, there is a large difference in availability of native and third-party libraries. Built-in and specialized functions and data types (e.g., float3) are also different between CUDA and OpenCL (and between Nvidia and AMD). If your code makes use of libraries or built-in functions and data types, the porting process will be more involved as substitutes must be found or implemented.

A difference of practical interest is that CUDA uses the (`nvcc`) compiler for compilation of the GPU code, whilst OpenCL uses the OpenCL driver. The CUDA compiler is a separate program that performs compilation of both CPU and GPU code and links these together into a single file. This process can take a significant amount of time, ranging from seconds to minutes depending on how complex the code is. OpenCL, on the other hand, uses the OpenCL driver for compilation, and this process is very fast in our experience. Both CUDA and OpenCL cache their compilations, so this only applies to the first time a kernel is being used. When developing GPU kernels, however, it becomes noticeably slower to work with the compilation times of PyCUDA.

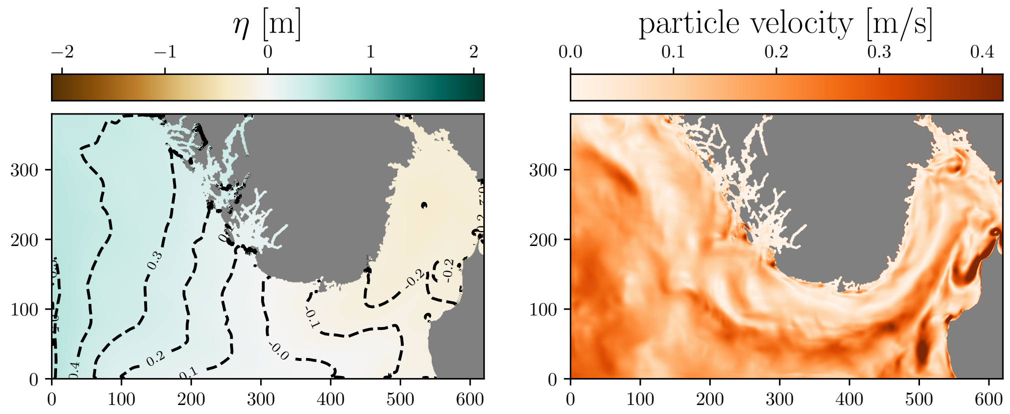


Figure 2: Example of a shallow water simulation for oceanographic purposes with our code. The left figure shows the sea-surface level, whereas the right figure shows the particle velocity. The simulation covers the North Sea around the southern part of Norway, with Denmark in the lower right corner. The axes shows distances in km, and the grid consists of 1550×950 cells of size $400\ \mathrm{m} \times 400\ \mathrm{m}$. The simulation here is initialized from operational data provided by the Norwegian Meteorological Institute.

4.2 Profile-Driven Optimization

The shallow-water equations consist of three coupled nonlinear partial differential equations that describe conservation of mass and momentum. In our case, they are formulated using η, the surface deviation from the equilibrium water level, and volume transport hu and hv along the abscissa and ordinate, respectively. Source terms represent varying bathymetry and the Coriolis forces, which takes into account that we solve the equations on a rotating sphere. The equations can be used to model gravitational waves, in which the governing motion is horizontal such as e.g., the ocean [7, 8]. An oceanographic simulation scenario using our simulator is shown in Figure 2.

We started the analysis of the three different numerical schemes using the Nsight extension in Visual Studio and the standalone Visual Profiler application in the exact same manner as if we were profiling from C/C++. NSight was run on a laptop with a GeForce 840M GPU in Windows, and the Visual Profiler on a desktop with a GTX 780 GPU in Linux. Our workflow started by profiling the code, identifying the performance bottleneck, optimizing the bottleneck, and finally profiling to determine if the optimization was successful [47]. To ensure that our optimizations do not introduce errors in the code, we frequently run integration and regression tests against reference solutions.

An important performance parameter for GPUs is the domain decomposition determined by the block size. CUDA decomposes the work into a grid with equally sized blocks, and all blocks are executed independently. At runtime, the GPU takes the set of blocks and schedules them to the different cores within the GPU. Using a too small block size will under-utilize the GPU, and using a too large block size will similarly exhaust the GPU's resources. Figure 3 shows how the block size has a major impact on performance for three different GPUs, and also illustrates that finding the best block size can be difficult. Because of this, we experimentally obtain the optimal configuration for each scheme before starting profiling, and again after the code has been optimized.

4.2.1 The High-Resolution Finite Volume Scheme

The first numerical scheme we consider is a high-resolution finite-volume method, which is designed to be well-balanced with respect to steady-state solutions in geostrophic balance [48]. The scheme reads the physical variables, (η, hu, hv), from GPU main memory, reconstructs an intermediate set of four geostrophic equilibrium variables, calculates inter-cell fluxes, and finally sums up the fluxes and writes the result back to GPU main memory. Each of the steps are stencil operations building on top of each other, and the kernel relies heavily on the use of shared memory for storing and

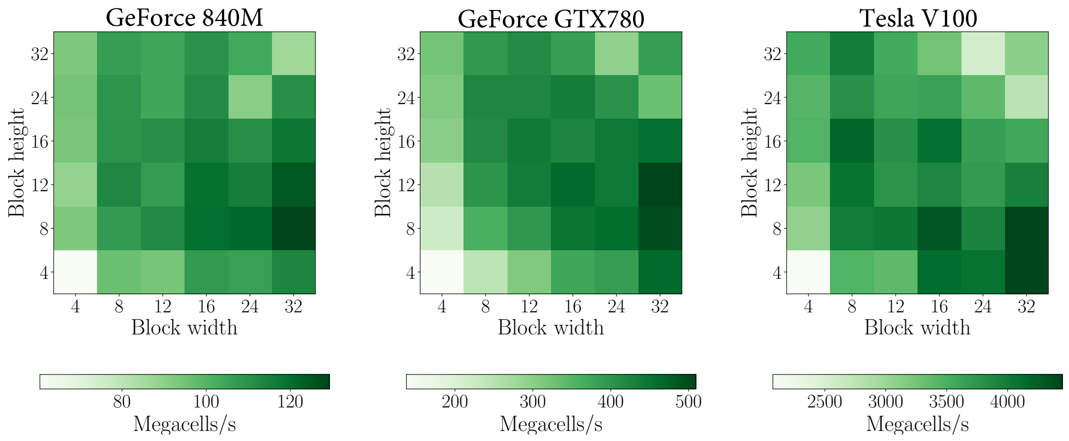


Figure 3: Heat map of performance as a function of block width and block height for selected sizes for the high-resolution scheme on three different GPUs. Notice that even though the performance patterns have similarities, the performance on the V100 would be suboptimal if the optimal configuration from the GTX780 is used. The performance increase is 2–5× for all three GPUs from the slowest to the fastest block size.

sharing intermediate results. The integration in time is based on a second-order strong stability preserving Runge-Kutta method, which means that the kernel is called twice for every iteration.

The first Nsight analysis of the high-resolution scheme indicates that the occupancy is very low at only 25%. The occupancy is a measure of how many threads are resident in the cores of the GPU simultaneously, and roughly translates to how well the GPU can hide memory latencies. As the GPU has restricted resources when it comes to the amount of shared memory, number of registers, etc., the occupancy is reduced if each block uses many of these resources. In our case, the limiting factor is the use of shared memory. By reducing the amount of shared memory used by each block, we can expect to get more blocks resident simultaneously on the GPU, which most likely will increase memory throughput and thereby increase performance.

Through six consecutive iterations, we progressively reduced the shared memory by 65% with the following steps (see Figure 4):

1. Recomputing bathymetry in cell intersections instead of storing H_m.
2. Recomputing face bathymetry instead of storing RH_x and RH_y.
3. Reusing buffer for physical variables Q for storing the reconstruction variables R.
4. Recomputing fluxes along the abscissa instead of storing F.
5. Recomputing fluxes along the ordinate instead of storing G.
6. Reusing the buffer for derivatives along the abscissa, Q_x, and derivatives along the ordinate, Q_y.

In the third and sixth steps we managed to reuse other shared memory buffers by reordering the execution flow of the code, and for all other cases we relied on re-computation. This essentially means that we prefer recomputing results rather than storing and sharing them between threads, thus trading extra computation for less memory storage.

The first seven groups in Figure 4 show the impact of the shared memory on performance. The first few buffers we removed were insufficient to increase the occupancy, because it would not free enough space for another resident block per core. However, after stage three there was space for one extra block, and the occupancy increased. After all six iterations occupancy increased to 56%, memory bandwidth utilization increased by a factor 1.8, and gigaFLOPS more than doubled. It should be noted that memory throughput is the important factor here, both since our kernel is memory

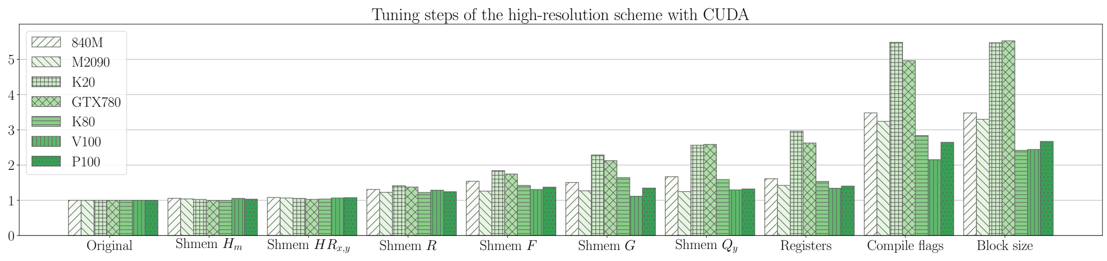


Figure 4: Different optimization stages for the high-resolution kernel showing performance normalized with respect to the original version. Notice that the different optimization strategies impact the performance very differently for different architectures. The platform we profiled on was the 840M GPU, yet the highest performance gain was for the K20 and GTX780 GPUs with over 5× improvement.

bound and because our re-computations artificially increase the number of floating point operations performed by the kernel by 20%.

The next limiting factor was the number of registers (variables) used by the kernel. First, we removed all variables related to debugging, and packed four boolean kernel arguments into a single integer using bit operations. Finally, we removed several temporary variables by aggregating these as soon as possible into the final computed flux. The compiler is already very good at optimizing register use, but using the above optimizations reduced the number of registers per thread from 49 to 47, which was sufficient to increase occupancy from 56% to 62%.

Compilation flags can also be used to increase performance with relatively little effort, as can be seen in Figure 4. Here, we used the `--use_fast_math` flag which enables using fast, albeit less precise, mathematical functions for operations such as exponential and square roots. This gave a dramatic effect and increased memory throughput by a factor of 3.9 relative to the original version. This is not only because the mathematical functions execute faster, but perhaps more importantly because the fast mathematical functions use less register space, reducing from 47 to 40 registers per thread. This increased occupancy to 69%, and left shared memory as the bottleneck once again. In our case, the use of fast mathematical functions is sufficiently accurate, but these compilation flags can ruin the correctness of a program and should be used with care. Other important flags to consider are flags that determine the maximum amount of register and specify cache configurations. After having optimized the kernel, our block size is probably no longer the optimal, and we rerun the block size optimization.

The performance increased by a factor 3.5 on the 840M, and 5.5 on the GTX 780 after these optimization steps. One surprising point is that the optimization steps have very different effect on different architectures. For example, the K20 and GTX780 increased to over 5 times the performance, whereas the K80, P100 and V100 increased by only half of that. Some steps also actually decrease the performance on one GPU, whilst having a positive effect on others. This was especially noticeable with our attempts at optimizing register use. It should still be noted that none of the optimizations steps that reduced shared memory usage led to a slowdown on any of the GPUs, even though these steps increased the computational workload per thread.

4.2.2 The Nonlinear Finite Difference Scheme

Our second scheme is a nonlinear second-order classical leapfrog scheme, also called centered-in-time centered-in-space. In this scheme, the three physical variables are defined on a staggered grid, and are updated using separate kernels. Each kernel reads all three variables from GPU main memory, performs computations, and writes back the result. The computational work in these three kernels is significantly less than for the high-resolution scheme, making this scheme even more memory bound. However, this also means that only four shared memory buffers are needed (three for the variables from the current time step and one for the constant depth). The initial profiling shows that the occupancy is 100% for all three kernels.

Due to its simplicity, this type of kernel leaves less headroom for optimization than the aforementioned high-

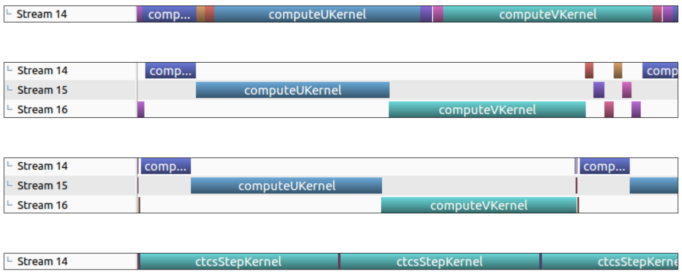


Figure 5: Four screenshots from the Nvidia Visual Profiler, illustrating the effect of optimizations for the nonlinear scheme. The top figure shows the original performance, and in the second we introduce multiple streams so that the three physical variables can be computed independently. In the third figure we reduce the number of blocks that are needed for the boundary condition kernels, and finally we compute all three variables within one kernel in the bottom figure. The horizontal time scale is about 7 ms and equal for all four figures, and shows that computing all variables in a single kernel gives the highest performance gain.

resolution scheme. For example, the compilation flag `--use_fast_math`, which had a dramatic effect for the high-resolution scheme, has only a marginal effect on this kernel because it performs less mathematical operations and uses few registers already. One thing that we can optimize, however, is to use concurrent kernel execution, since all three variables can be updated simultaneously. Unfortunately, each kernel already occupies all available resources on the GPU, and therefore the effect turns out to be marginal, as shown in Figure 5. The figure also shows that the boundary condition kernels took too long time, and directed our attention to them. These could with relative ease be optimized to take an insignificant amount of time.

Because we launch three kernels to update the variables, we read the variables η, hu and hv multiple times from GPU main memory for every iteration. By carefully gathering these kernels into one, we can reduce the number of variables read and written from GPU main memory from 16 to ten. After this optimization, our profiling shows a 36% decrease in memory traffic, compared to the theoretical 37.5%. This directly translates to an equivalent improvement in performance, as shown in the lower panel of Figure 5.

By merging the three kernels into one, the occupancy decreased to 62.5%, as each thread now requires more registers. Several attempts were made to reduce this number, but none of our attempts helped the compiler to do a better job. One way of forcing fewer registers is to set the `--maxrregcount` compilation flag. However, this implies that registers are spilled to local memory (cache on the GPU), and even though the occupancy increases, the actual performance decreases. The final results from optimizing the nonlinear scheme is shown in Figure 6, and the optimizations increase performance of the CUDA versions by between two to three times on all seven GPUs.

4.2.3 The Linear Finite Difference Scheme

The final scheme solves the linearized shallow-water equations using a forward-backward linear finite-difference scheme [49] and is asymmetric in time. It consists of three simple kernels, in which the most recent results are always used (similarly to Gauss-Seidel iteration):

$$\begin{aligned} hu^{n+1} &\leftarrow F(\eta^n, hv^n, hu^n), \\ hv^{n+1} &\leftarrow G(\eta^n, hv^n, hu^{n+1}), \\ \eta^{n+1} &\leftarrow H(\eta^n, hv^{n+1}, hu^{n+1}). \end{aligned}$$

In order to reduce the amount of memory read from GPU main memory, we apply the same strategy as for the nonlinear scheme and carefully combine all three kernels into one. Since the execution order has to comply with the data dependencies, we need to read extra input data (referred to as ghost cells or computational halo) for all variables at time step n. We then compute hu^{n+1}, hv^{n+1}, and finally η^{n+1}. Our profiling shows that this reduced the amount of memory read and written to GPU main memory by 50%.

The profiling now tells us that memory dependencies are the main bottleneck, and that the GPU has no eligible instructions for about 60% of the cycles. Without a major redesign of the algorithm, there is typically little we can do to improve the kernel further, but for completeness we added the `--use_fast_math` compilation flag. However, since the kernel is simple and heavily memory bound the flag had a negligible impact.

The overall performance increase is more than two times for the laptop 840M GPU and the server V100 GPU, but only around 40% for the others. It should also be noted that this kernel is the one with the smallest negative performance impact when moving from OpenCL to CUDA.

4.3 Backporting Optimizations to OpenCL

To do a fair comparison of the performance of PyCUDA and PyOpenCL, we need to back-port our optimizations to PyOpenCL. As most of the optimizations are carried out in the CUDA source code, it is a relatively simple matter of copying the optimized code into the original OpenCL kernels, and update the code according to Table 3. In the Python code, the loading of the separate kernels for the linear and nonlinear schemes are replaced by the new merged kernels, and the input arguments are updated to correspond to these optimizations. Active and appropriate use of a version control system and regression testing is crucial in this work.

Porting compiler flags can in general be a greater challenge, as there are no one-to-one overlap between the compile-time options for the two languages. However, in our case only the `--use_fast_math` compilation flag is used in the final PyCUDA version, and it has a matching counterpart, `-cl-fast-relaxed-math`, that is passed to PyOpenCL's API when compiling the kernels.

4.4 Comparing Performance

The overall performance gain of our optimization is shown in Figure 6, where all results are given in megacells per second normalized with respect to the original PyOpenCL implementation in the left-hand column and GPU bandwidth (see Table 2) in the right-hand column. The original porting from PyOpenCL to PyCUDA gave a noticeable reduction in performance for the high-resolution scheme on all GPUs. After careful examination, we attribute this to different default compilation flags in PyCUDA and PyOpenCL: In PyCUDA, the fast-math flag was shown to double the performance for the high-resolution scheme, while we found that it gave less than 5% performance gain with PyOpenCL. Note that the slowdown in the original porting is much less for the linear and nonlinear schemes, as these schemes contain fewer complex mathematical operations, and we instead observe a varying effect on performance of porting the original OpenCL code to CUDA. When examining the numerical schemes one by one, we see that the optimizations performed for the high-resolution scheme appears to be highly portable when back-ported to PyOpenCL for all GPUs. For the tuned nonlinear scheme, however, we see that the 840M and V100 GPUs give significantly higher performance using CUDA than OpenCL. Finally, for the linear scheme, the performance is similar for all GPUs, and only the 840M and V100 GPUs benefit significantly from the optimization effort. In total, we see that certain scheme and GPU combinations result in a significant speedup for CUDA over OpenCL, but we cannot conclude whether this is caused by differences in driver versions or from other factors. We are therefore not able to claim that CUDA performs better than OpenCL in general. When looking at the performance normalized with respect to the specific GPU bandwidth, we see that the 840M laptop GPU offers the highest performance for all schemes, followed by the two most recent high-end GPUs, the V100 and P100.

4.5 Measuring Power Consumption

We measure power consumption in two ways. The first method is by using the nvidia-smi application, which can be used to monitor GPU state parameters such as utilization, temperature, power draw, etc. By programmatically running nvidia-smi in the background during benchmark experiments, we can obtain a log containing a high-resolution power draw profile for the runtime of the benchmark. The downside of using nvidia-smi is that information about power draw is only supported on recent high-performance GPUs, and we have therefore only benchmarked the power consumption with nvidia-smi on the two most recent Tesla GPUs, the P100 and V100. Further, nvidia-smi monitors the energy consumption of the GPU only, meaning that we do not have any information about energy consumed by the CPU. For each experiment, nvidia-smi is started in the background exactly 3 s before the benchmark, and is configured to log the

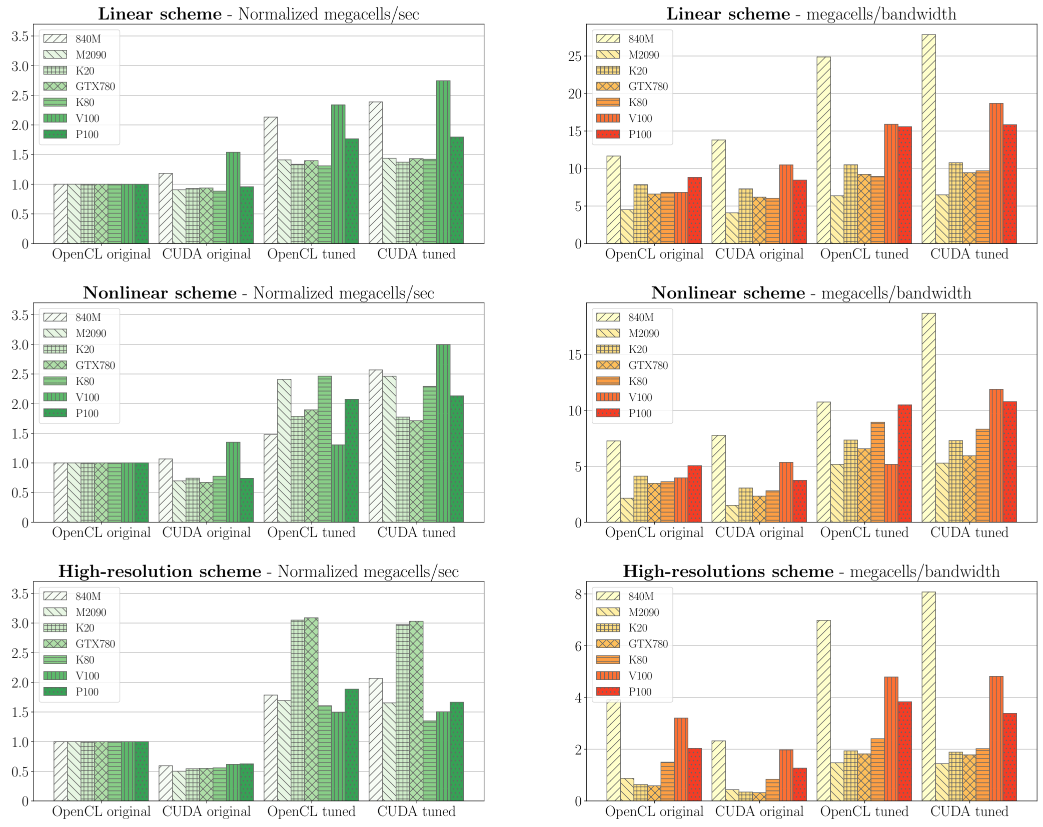


Figure 6: Performance of original, ported, and optimized kernels measured in megacells per second. The left column is normalized with respect to original performance, and the right column with respect to the theoretical GPU bandwidth. Notice that there is relatively little difference between CUDA and OpenCL, whilst there is a significant difference in how effective the tuning is for the different architectures. Furthermore, there is a significant loss of performance when porting from OpenCL to CUDA in our original approach for the high-resolution and nonlinear schemes. From our experience, this relates to how the two languages optimize mathematical expressions with and without the fast math compilation flags. Also notice that the 840M, V100 and P100 GPUs achieve the best performance relative to the GPU bandwidth.

power draw every 20 ms. This background process is stopped again exactly 3 s after the end of the simulation. This approach allows us to measure the energy consumption of the idle GPU both before and after each benchmark, and we ignore the idle sections when computing the mean and total power consumption for each experiment. All results presented here are with the idle load subtracted from the experiment results.

The second method is to measure the total amount of energy used by the entire computer through a watt meter. The use of the watt meter requires physical access to the computer, and we are therefore restricted to do measurements on the laptop and desktop, containing the GeForce 840M and GeForce GTX780 GPUs, respectively. The watt meter offers no automatic logging or reading, but displays the total power used with an accuracy of 1 Wh. To get sufficiently accurate readings we need to run each benchmark long enough to keep the GPU busy for approximately one hour, after which we read the total and mean power consumption for each experiment. Before and after each benchmark, we also record the background power of the idle system, and the maximum recorded power during the experiment, to monitor whether the operating system is putting any non-related background load on the computer. It should also be noted that the battery was removed from the laptop during these experiments. Similarly to the first method, we subtract the idle loads from the result of each experiment, but note that the addition increased load from the CPU will still be included.

4.6 Comparing Energy Efficiency

Figure 7 shows the results from the power efficiency experiments using the watt meter on the laptop (840M) and desktop (GTX780). The top row repeats the results for computational performance also shown in Figure 6 for the relevant GPUs, whereas the second row show the normalized mean power consumption with respect to the original OpenCL versions. The first thing we notice is that CUDA seems to require less power on the 840M compared to OpenCL for all versions of the three schemes. On the GTX780, however, there are no differences between the two programming models for equivalent versions. In fact, only the tuned high-resolution scheme seems to be different from the others (using about 30% more power), and this behavior can also be seen on the 840M. The power efficiency of the three schemes is shown in the bottom row in Figure 7, and we see that on the 840M the tuned CUDA versions are the most power efficient. This is because CUDA is both more efficient and uses less power on this system. On the GTX780, CUDA and OpenCL have equivalent power efficiency for all tuned schemes.

The results for the Tesla P100 and Tesla V100 are shown in Figure 8. The top row shows the computational performance in megacells per second, repeating the results from Figure 6. The second row shows the mean power used by each version of the three schemes. Note here that on the V100, both CUDA versions for the nonlinear scheme use 60–90% more power than the OpenCL versions, which is the opposite result compared to the 840M results. For both the linear and high-resolution schemes, the results do not differ significantly in favor of either CUDA or OpenCL, but the tuned OpenCL version uses slightly more power for the high-resolution scheme. When we consider the power efficiency in the bottom row, we see that the tuned CUDA versions are the best versions for all schemes. In particular for the nonlinear scheme, this is mostly due to the large difference in computational efficiency between CUDA and OpenCL for this particular scheme on this particular GPU. For the P100, however, the mean power consumption is almost the same for all versions of each scheme, except for the high-resolution scheme which has an increased mean power consumption for the tuned versions. The increase in mean power consumption is however less than the increase in computational performance, and the high-resolution scheme also has a higher energy efficiency when tuned.

In general, we observe that all experiments show a mean power usage within about 30% of the original OpenCL versions, with the exception of the nonlinear scheme on the V100. On the other side, the computational performance increases up to 5 times (the high-resolution scheme on the GTX780). This shows that the most important factor for improving power efficiency is to increase computational performance.

4.7 Tuning Block Size Configuration for Energy Efficiency

In the previous section, all benchmarks were configured using the optimal block size configuration for computational efficiency. Here, we analyze how sensitive power efficiency is to the block size configuration, and whether there is other optimal configurations if we aim for power efficiency instead of computational performance.

Figure 9 shows benchmark results for the three different schemes for a wide range of block size configurations. The top row shows computational efficiency, in terms of computed megacells per second, and confirms how sensitive the performance for all three schemes are to different configurations, similarly to the results from Figure 3. The

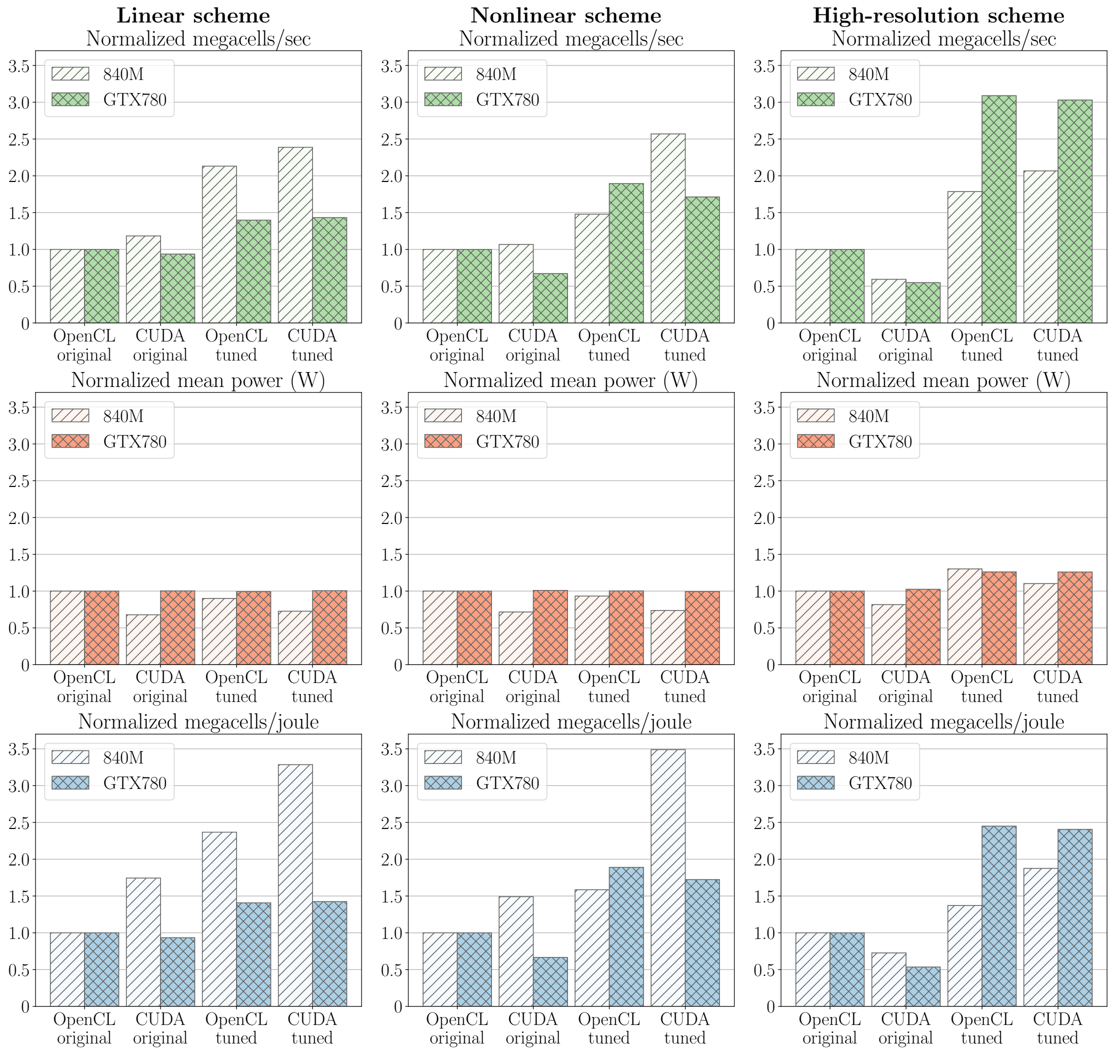


Figure 7: Comparison of original, ported and optimized codes measured in megacells per second (top row), mean power usage (mid row), and megacells per joule (bottom row), normalized with respect to the original OpenCL implementation, for the laptop (840M) and desktop (GTX780) GPUs. The power is measured through a watt meter, and represents the power consumed by the entire computer. Note that the CUDA versions requires less power than the OpenCL versions on the 840M, whereas there are no differences between equivalent versions on the GTX780. In terms of power efficiency, CUDA is more efficient than OpenCL on the M840, whereas the GTX780 gives the same power efficiency.

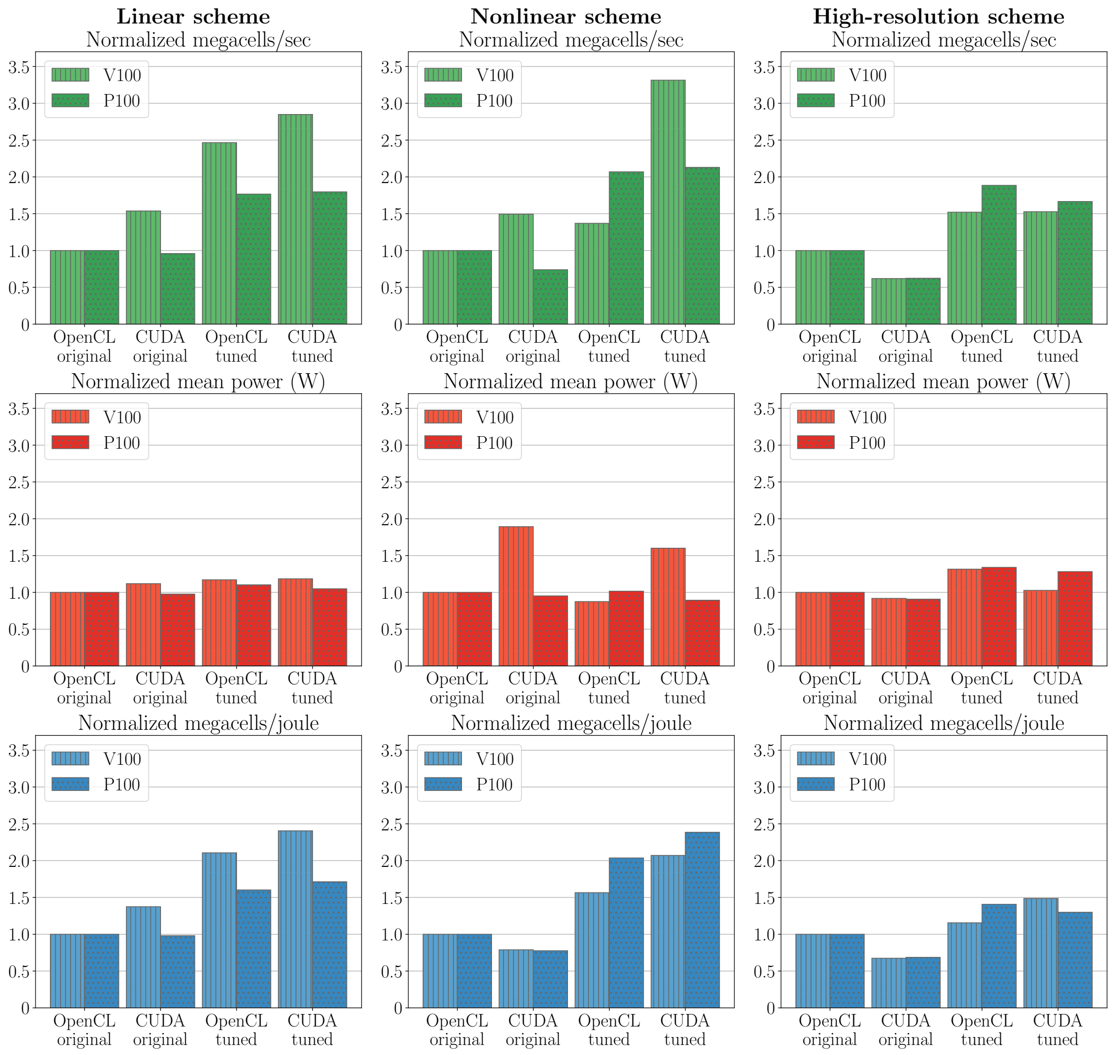


Figure 8: Comparison of original, ported and optimized codes measured in megacells per second (top row), mean power usage (mid row), and megacells per joule (bottom row), normalized with respect to the original OpenCL implementation, for the Tesla P100 and V100 GPUs. The power is measured through nvidia-smi, and represents the power consumed by the GPU only. On the V100, there are only minor differences in mean power consumption between different versions of the linear and high-resolution scheme, but CUDA uses more power than OpenCL for the nonlinear scheme. CUDA is, however, more power efficient than OpenCL for all three tuned schemes on this GPU. On the P100, the mean power consumption is equal for all versions of the linear and nonlinear schemes, which means that power efficiency becomes equivalent to computational efficiency. For the high-resolution scheme we see that the mean power consumption increases for the tuned codes, but the computational performance increases more, meaning that the power efficiency is still improved by 30–40%.

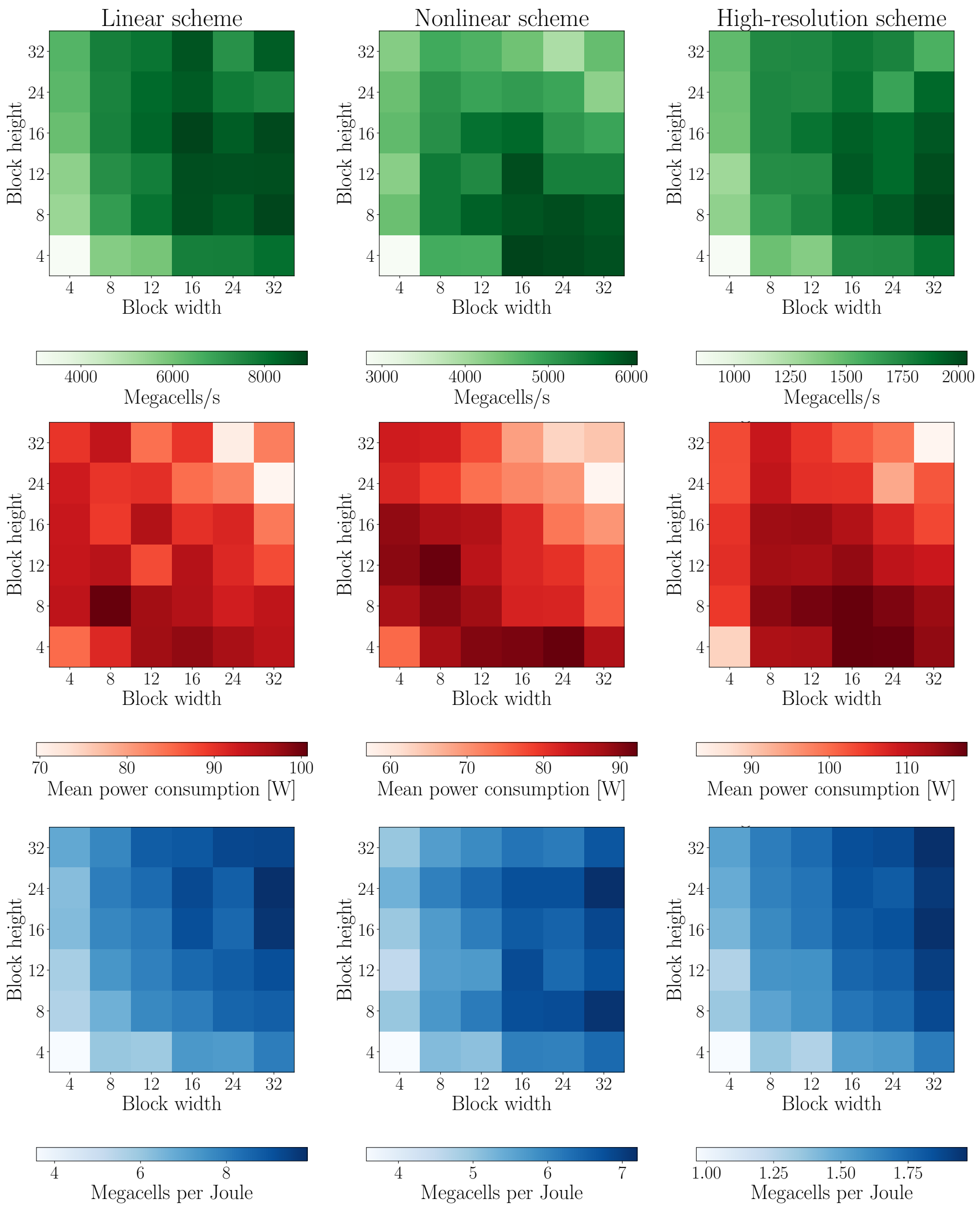


Figure 9: Computational performance (top), mean power consumption (middle), and energy efficiency (bottom) for the three numerical schemes for different block size configurations on the Tesla P100 GPU. Notice that all three measures change rapidly and unpredictably between the different configurations, and that there does not seem to be a direct translation between high computational performance and the mean power consumption. The bottom row shows that it is just as important to tune for optimal block size configuration when optimizing for energy efficiency as when optimizing for computational performance, and that these two objectives cannot be expected to be fulfilled simultaneously.

optimal configurations with respect to computational efficiency are $(16, 16)$, $(16, 4)$ and $(32, 8)$ for the linear, nonlinear and high-resolution scheme, respectively. The second row shows the mean power consumption for each execution. By comparing the heat map of mean power usage with the computational efficiency, we see that high computational performance does not necessarily translate into high mean power consumption. As a specific example, the most efficient configuration for the linear scheme has a lower mean energy consumption than half of the configurations. The final row of Figure 9 shows energy efficiency with respect to block size configuration, and again we see that the results do not follow an easily predictable pattern across the different block sizes. Notice how the most energy efficient block sizes ($(32, 24)$ for the linear and nonlinear scheme, and $(32, 32)$ for the high-resolution scheme) can be recognized as configurations that have particularly low mean power consumption and are less than medium fast. None of the most computational efficient configurations are among the very best configurations in terms of energy efficiency. These results show that block tuning for computational efficiency and for energy efficiency are not the same. However, because a performance tuned code has a lower time-to-solution, we can in general claim that performance tuning increases the power efficiency proportionally to performance.

5 Summary

We have presented our experiences from working with CUDA and OpenCL from Python for an extensive period of time[6]. Our conclusion is that using Python is as computationally efficient as C++ for our use, and we believe this to be true for many other application areas as well. We have also benchmarked three different OpenCL codes, our ported code in CUDA, our optimized CUDA code, and finally our OpenCL code with optimizations found using the CUDA tools. Our results are shown for seven different GPUs, thus representing many of the GPU architectures in use today. Finally, we have looked at the power consumption for all versions of the code and the potential for tuning for energy efficiency.

One of our most important findings is that working with Python is significantly faster than using C++, as we have previously done. We notice that the amount of research we are able to do in the same amount of time increases dramatically, and that the code quality is higher with fewer bugs and crashes. The combination of PyCUDA/PyOpenCL and the Jupyter Notebook led to a very productive development environment after addressing the initial crashes that forced us to reboot often. For our application area, the overhead of using Python becomes negligible with respect to performance.

The original motivation for using OpenCL was to support GPUs and similar architectures from multiple vendors. Our motivation for changing from OpenCL to CUDA was because of the better software ecosystem for CUDA, and we have been very happy with our decision. CUDA appears to be a much more stable and mature development ecosystem with better tools for development, debugging and profiling for our hardware.

We found it interesting that our initial port from OpenCL to CUDA imposed a performance penalty, due to different default compiler optimizations. Even though some authors have reported OpenCL to be slower than CUDA, we find no conclusive results that support this in general. The performance gain varied much more with the GPU being used than whether we used CUDA or OpenCL. In most of our experiments, tuned versions of CUDA and OpenCL was found to have the same computational performance, with a few examples showing CUDA being faster than OpenCL. Additionally, we found that the performance gain of a single optimization strategy will have vastly different effects on the run time for different GPUs. Even though we profiled and optimized mainly using a laptop GPU, the highest relative performance gain was for a server class and a desktop class GPU (the high-resolution scheme on the GTX780 and K20).

There does not seem to be any clear relationships between the power consumption when comparing different schemes, optimization levels, GPUs, and programming models. When we consider power efficiency, we see that CUDA performs better than OpenCL for all tuned schemes on the Tesla V100 and GeForce 840M GPUs, whereas there are much smaller differences on the GeForce GTX780 and Tesla P100 GPUs. When we examine the impact of performance tuning on power efficiency, there appears to be a systematic and clear relationship: A fast code is a power efficient code.

In terms of mean power consumption we find no clear relationships when comparing different schemes, optimization

[6]The authors have worked with GPU computing from C++ for over ten years, and over four years from Python

levels, GPUs, or programming models, but it seems to be just as important to consider block size configurations, as we measure a factor two difference between the lowest and highest mean power for different configurations. We also see that optimizing block size for power efficiency is not necessarily the same as optimizing for computational efficiency.

Acknowledgement

This work is supported by the Research Council of Norway through grant number 250935 (GPU Ocean). The Tesla K20 computations were performed on resources provided by UNINETT Sigma2 – the National Infrastructure for High Performance Computing and Data Storage in Norway under project number nn9550k.

Author Contributions

The authors have all contributed to all aspects related to the work, including methodology, software, benchmarking and preparing the manuscript. The authors have contributed close to equally to most parts of the paper, but H.H.H. has had a lead role in the aspect directly related to power efficiency, and has had the role of lead and corresponding author.

Conflicts of Interest

The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

[1] H. Holm, A. Brodtkorb, and M. Sætra. Performance and energy efficiency of CUDA and OpenCL for GPU computing using Python. Advances in Parallel Computing, 36:593–604, 2020.

[2] E. Larsen and D. McAllister. Fast matrix multiplies using graphics hardware. In Proceedings of the 2001 ACM/IEEE Conference on Supercomputing, SC '01, pages 55–55. ACM, 2001.

[3] J. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. Lefohn, and T. Purcell. A survey of general-purpose computation on graphics hardware. Computer Graphics Forum, 26(1):80–113, 2007.

[4] S. Nanz and C. Furia. A comparative study of programming languages in Rosetta Code. In IEEE International Conference on Software Engineering, volume 1, pages 778–788, 2015.

[5] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih. PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation. Parallel Computing, 38(3):157–174, 2012.

[6] K. Asanović, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams, and K. Yelick. The landscape of parallel computing research: A view from Berkeley. Technical report, EECS Department, University of California, Berkeley, 2006.

[7] A. Brodtkorb. Simplified ocean models on the GPU. In 2018: Norsk Informatikkonferanse, 2018.

[8] H. Holm, A. Brodtkorb, G. Broström, K. Christensen, and M. Sætra. Evaluation of selected finite-difference and finite-volume approaches to rotational shallow-water flow. Communications in Computational Physics, 27(4):1234–1274, 2020.

[9] T. Hagen, M. Henriksen, J. Hjelmervik, and K-A. Lie. How to Solve Systems of Conservation Laws Numerically Using the Graphics Processor as a High-Performance Computational Engine, pages 211–264. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[10] A. Brodtkorb, M. Sætra, and M. Altinakar. Efficient shallow water simulations on GPUs: Implementation, visualization, verification, and validation. Computers & Fluids, 55(0):1–12, 2012.

[11] M. de la Asunción, J. Mantas, and M. Castro. Simulation of one-layer shallow water systems on multicore and CUDA architectures. The Journal of Supercomputing, 58(2):206–214, 2011.

[12] A. Brodtkorb and M. Sætra. Explicit shallow water simulations on GPUs: Guidelines and best practices. In XIX International Conference on Computational Methods for Water Resources, pages 17–22, June 2012.

[13] J. Holewinski, L-N. Pouchet, and P. Sadayappan. High-performance code generation for stencil computations on GPU architectures. In Proceedings of the 26th ACM International Conference on Supercomputing, ICS '12, pages 311–320. ACM, 2012.

[14] OpenACC-Standard.org. The OpenACC application programming interface version 2.7, 2018.

[15] NVIDIA. NVIDIA CUDA C programming guide version 10.1, 2019.

[16] Khronos OpenCL Working Group. The OpenCL specification v. 2.2, 2018.

[17] A. Brodtkorb, C. Dyken, T. Hagen, J. Hjelmervik, and O. Storaasli. State-of-the-art in heterogeneous computing. Scientific Programming, 18(1):1–33, 2010.

[18] S. Huang, S. Xiao, and W-C. Feng. On the energy efficiency of graphics processing units for scientific computing. In 2009 IEEE International Symposium on Parallel & Distributed Processing, pages 1–8. IEEE, 2009.

[19] Z. Qi, W. Wen, W. Meng, Y. Zhang, and L. Shi. An energy efficient opencl implementation of a fingerprint verification system on heterogeneous mobile device. In 2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications, pages 1–8. IEEE, 2014.

[20] T. Dong, V. Dobrev, T. Kolev, R. Rieben, S. Tomov, and J. Dongarra. A step towards energy efficient computing: Redesigning a hydrodynamic application on CPU-GPU. In 2014 IEEE 28th International Parallel and Distributed Processing Symposium, pages 972–981. IEEE, 2014.

[21] V. Klôh, D. Yokoyama, A. Yokoyama, G. Silva, M. Ferro, and B. Schulze. Performance and energy efficiency evaluation for HPC applications in heterogeneous architectures. 2018.

[22] S. Memeti, L. Li, S. Pllana, J. Kołodziej, and C. Kessler. Benchmarking OpenCL, OpenACC, OpenMP, and CUDA: programming productivity, performance, and energy consumption. In Proceedings of the 2017 Workshop on Adaptive Resource Management and Scheduling for Cloud Computing, pages 1–6. ACM, 2017.

[23] P. Du, R. Weber, P. Luszczek, S. Tomov, G. Peterson, and J. Dongarra. From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming. Parallel Computing, 38(8):391–407, 2012.

[24] J. Fang, A. Varbanescu, and H. Sips. A comprehensive performance comparison of CUDA and OpenCL. In 2011 International Conference on Parallel Processing, pages 216–225, 2011.

[25] T. Gimenes, F. Pisani, and E. Borin. Evaluating the performance and cost of accelerating seismic processing with CUDA, OpenCL, OpenACC, and OpenMP. In 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pages 399–408, 2018.

[26] K. Karimi, N. Dickson, and F. Hamze. A performance comparison of CUDA and OpenCL. Technical report, D-Wave Systems Inc., 2011.

[27] G. Martinez, M. Gardner, and W c. Feng. CU2CL: A CUDA-to-OpenCL translator for multi- and many-core architectures. In 2011 IEEE 17th International Conference on Parallel and Distributed Systems, pages 300–307, 2011.

[28] J. Kim, T. Dao, J. Jung, J. Joo, and J. Lee. Bridging OpenCL and CUDA: a comparative analysis and translation. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pages 1–12, 2015.

[29] D. Kaeli, P. Mistry, D. Schaa, and D. Zhang. Heterogeneous Computing with OpenCL 2.0. 2015.

[30] J. Sanders and E. Kandrot. CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley Professional, 2010.

[31] University of Mannheim, University of Tennessee, and NERSC/LBNL. Top 500 supercomputer sites. `http://www.top500.org`, June 2019.

[32] AMD Developer Tools Team. CodeXL quick start guide, 2018.

[33] Intel. Intel SDK for OpenCL applications, 2018.

[34] Intel. Intel VTune Amplifier, 2018.

[35] Apple Inc. Metal programming guide, 2018.

[36] G. Wilson, D. Aruliah, C. Brown, N. Chue Hong, M Davis, R. Guy, S. Haddock, K. Huff, I. Mitchell, M. Plumbley, B. Waugh, E. White, and P. Wilson. Best practices for scientific computing. PLOS Biology, 12(1):1–7, 2014.

[37] L. Prechelt. An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program. Technical report, Karlsruhe Institute of Technology, 2000.

[38] S. Lam, A. Pitrou, and S. Seibert. Numba: A LLVM-based python JIT compiler. In Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC, LLVM '15, pages 7:1–7:6. ACM, 2015.

[39] S. Behnel, R. Bradshaw, and D. Seljebotn. Cython tutorial. In Gaël Varoquaux, Stéfan van der Walt, and Jarrod Millman, editors, Proceedings of the 8th Python in Science Conference, pages 4–14, Pasadena, CA USA, 2009.

[40] A. Kaehler and G. Bradski. Learning OpenCV: Computer Vision in C++ with the OpenCV Library. 2017.

[41] R. Okuta, Y. Unno, D. Nishino, S. Hido, and C. Loomis. CuPy: A NumPy-compatible library for NVIDIA GPU calculations. In Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS), 2017.

[42] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, and Jupyter development team. Jupyter notebooks – a publishing format for reproducible computational workflows. In Positioning and Power in Academic Publishing: Players, Agents and Agendas, pages 87–90, 2016.

[43] J. Hunter. Matplotlib: A 2d graphics environment. Computing In Science & Engineering, 9(3):90–95, 2007.

[44] R. Singh, P. Wood, R. Gupta, S. Bagchi, and I. Laguna. Snowpack: Efficient parameter choice for GPU kernels via static analysis and statistical prediction. In Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, pages 8:1–8:8, New York, NY, USA, 2017. ACM.

[45] J Price and S. McIntosh-Smith. Analyzing and improving performance portability of OpenCL applications via auto-tuning. In Proceedings of the 5th International Workshop on OpenCL, pages 14:1–14:4, New York, NY, USA, 2017. ACM.

[46] T. Falch and A. Elster. Machine learning-based auto-tuning for enhanced performance portability of OpenCL applications. Concurrency and Computation: Practice and Experience, 29(8), 2017.

[47] A. Brodtkorb, T. Hagen, and M. Sætra. Graphics processing unit (GPU) programming strategies and trends in GPU computing. Journal of Parallel and Distributed Computing, 73(1):4–13, 2013.

[48] A. Chertock, M. Dudzinski, A. Kurganov, and M. Lukácová-Medvidová. Well-balanced schemes for the shallow water equations with Coriolis forces. Numerische Mathematik, Dec 2017.

[49] A. Sielecki. An energy-conserving difference scheme for the storm surge equations. Monthly Weather Review, 96(3):150–156, 1968.

Paper III

Massively Parallel Implicit-Equal Weights Particle Filter for Ocean Drift Trajectory Forecasting

Håvard Heitlo Holm, Martin Lilleeng Sætra, Peter Jan van Leeuwen

Journal of Computational Physics: X, volume 6, number 100053, 2020.
DOI: 10.1016/j.jcpx.2020.100053

Massively Parallel Implicit Equal-Weights Particle Filter for Ocean Drift Trajectory Forecasting

Håvard Heitlo Holm*[1,2], Martin Lilleeng Sætra[3,4], and Peter Jan van Leeuwen[5,6]

[1] Mathematics and Cybernetics, SINTEF Digital, P.O. Box 124 Blindern, NO-0314 Oslo, Norway.
[2] Department of Mathematical Sciences, Norwegian University of Science and Technology, NO-7491 Trondheim, Norway.
[3] Information Technology Department, Norwegian Meteorological Institute, P.O. Box 43 Blindern, NO-0313 Oslo, Norway.
[4] Department of Computer Science, Oslo Metropolitan University, P.O. Box 4 St. Olavs plass, NO-0130 Oslo, Norway.
[5] Department of Atmospheric Science, Colorado State University, 3915 W. Laporte Ave. Fort Collins, CO 80521, USA
[6] Department of Meteorology, University of Reading, Earley Gate, Reading RB6 6BB, UK.

Abstract

Forecasting of ocean drift trajectories is important for many applications, including search and rescue operations, oil spill cleanup and iceberg risk mitigation. In an operational setting, forecasts of drift trajectories are produced based on computationally demanding forecasts of three-dimensional ocean currents. Herein, we investigate a complementary approach for shorter time scales by using the recently proposed two-stage implicit equal-weights particle filter applied to a simplified ocean model. To achieve this, we present a new algorithmic design for a data-assimilation system in which all components – including the model, model errors, and particle filter – take advantage of massively parallel computing architectures, such as graphical processing units. Faster computations can enable in-situ and ad-hoc model runs for emergency management, and larger ensembles for better uncertainty quantification. Using a challenging test case with near-realistic chaotic instabilities, we run data-assimilation experiments based on synthetic observations from drifting and moored buoys, and analyse the trajectory forecasts for the drifters. Our results show that even sparse drifter observations are sufficient to significantly improve short-term drift forecasts up to twelve hours. With equidistant moored buoys observing only 0.1% of the state space, the ensemble gives an accurate description of the true state after data assimilation followed by a high-quality probabilistic forecast.

1 Introduction

Prediction of drift trajectories in the ocean has many applications that are important to society and the environment. Examples include search and rescue operations, recovering objects lost at sea, planning of boom placements for oil spill cleanup, and preventing collisions between icebergs and offshore installations. To produce high-quality drift trajectory forecasts, it is important to have a good representation of ocean currents. This is not an easy task, as ocean currents have large natural variability and there are typically few available observations. Furthermore, the size of ocean low- and high-pressure systems, so-called *eddies*, is much smaller than their atmospheric counterparts, and it is challenging to place them correctly in typical grid resolutions used by operational ocean models today.

The operational approach for drift trajectory prediction is to use the currents from the most recent ocean forecasts directly [1]. These are imported from computationally expensive ocean circulation models, such as ROMS [2], which solve the dynamic state of the ocean in three dimensions. Typically, a large portion of the simulation run-time is spent on the data assimilation, which uses available real-world observations to correct the modeled ocean states that serve as initial conditions for the next forecast. Common forecast ranges for ocean circulation models are three to five days. Operational drift trajectory forecasts at the Norwegian Meteorological Institute (MET Norway) are produced by

*Corresponding author: havard.heitlo.holm@sintef.no

OpenDrift [1], which is an offline trajectory model that reads the ocean current forecasts to predict drift trajectories. Although OpenDrift is computationally efficient, the ocean circulation models still require access to supercomputers.

This paper explores the option of using a recently proposed particle filter method applied to a simplified ocean model for efficient drift trajectory forecasting. The aim is to build a data-assimilation system that can run efficiently on commodity-level desktop computers, and also be extendable to supercomputers. We achieve this by using a simplified ocean model and a data-assimilation method that both are able to take advantage of massively parallel accelerator hardware, such as the graphical processing unit (GPU). This work is not intended as a substitute of current operational systems, but as a complementary approach, in which the predicted currents may even be updated with in-situ observations, e.g., during ongoing search and rescue operations. Furthermore, by enabling research models to run on individual desktop and laptop computers, researchers are able to do more rapid prototyping. At the same time, this work will contribute to more efficient simulations also on supercomputers, since all algorithms may be extended to run on multiple GPUs and compute nodes.

The paper is organized as follows: We start by highlighting our contributions and reviewing related work relevant for Lagrangian data assimilation with accelerated particle filters. In Section 2, we describe the data-assimilation problem and summarize the key concepts of so-called proposal-distribution particle filters. We present the simplified ocean model and model errors in Section 3, whereas Section 4 offers a detailed description of an algorithm for running the chosen particle filter on this model. The latter two sections also discuss how the GPU is used for efficient implementation of the computationally intensive components. In Section 5, we present numerical experiments of drift trajectory ensemble forecasts, using an identical-twin experiment setup designed to resemble real-world ocean currents and with configuration inspired by operational systems. Furthermore, we show and discuss the statistical validity of the forecasts and examine the computational performance of the simulations. Finally, Section 6 contains a summary and concluding remarks.

Paper contribution We present an efficient GPU-accelerated data-assimilation system based on the recently proposed implicit equal-weights particle filter (IEWPF) applied to simplified ocean models described by the rotating shallow-water equations. The data-assimilation algorithm, the numerical scheme for evolving the ocean model, and the algorithm for sampling locally correlated, well-balanced random model errors are all designed to take advantage of massively parallel architectures. The data-assimilation system is tailored for observations of the ocean current obtained from either free drifting buoys or moored buoys. We show numerical experiments for assimilating a challenging test case with near-realistic chaotic behavior, along with drift trajectory forecasts. The results show that by assimilating approximately 0.1% of the state space, the posterior ensemble mean strongly resembles the true state in the entire domain, thus enabling an accurate drift trajectory forecasts. This is also the first time that the IEWPF method is applied to high-dimensional geophysical problems. Furthermore, we show that the particle filter has well-behaved statistical properties, and that the computational efficiency of the data assimilation is well-balanced with respect to the model. To the best of our knowledge, there exists no previous massively parallel implementations of a state-of-the-art particle filter applied to a high-dimensional geophysical system.

Related work Particle filters, and more generally Sequential Monte Carlo (SMC) methods, constitute a large class of numerical methods for statistical inference. It is well-known that the standard particle filter is prone to degeneracy in high-dimensional systems [3, 4, 5], and there have been several attempts at designing particle filters without this limitation. A few such particle filters have been used on high-dimensional, near-realistic applications in the geosciences. Ades and van Leeuwen [6] use the equivalent-weights particle filter on a high-dimensional, simplified, ocean model based on the barotropic equations, showing that it is possible to avoid the degeneracy problem in high-dimensional systems, at the cost of a biased estimate. Although the scheme performed well, the bias grows with ensemble size. Poterjoy, Sobash and Anderson [7] use a local particle filter on a weather research and forecasting model, in which the bootstrap particle filter is applied locally to observations, and particle states are merged in state space between the locations of the observations. However, it remains problematic to glue particles from these local updates together to full particles that span the whole model domain. The smoothing needed can easily destroy delicate balances in the flow. Furthermore, the minimum size of the local areas is set by physical length-scale constraints, typically meaning that too many observations are within a local domain to avoid degeneracy. In practise, a minimum weight value is set, meaning that not all information is extracted from the observations. Hence, also localisation is not solving the problem.

A recent review by van Leeuwen et al.[8] discusses most recent developments on particle filters for high-dimensional geophysical systems.

Several implementations of standard particle filters for parallel architectures such as GPUs exist, but mainly within other scientific disciplines than geosciences. Lopez et al. [9] present GPU-implementations of a particle filter (with sequential importance resampling) and auxiliary particle filter to detect anomalies in manufacturing processes, and show sufficient performance for real-time application. Gelencsér-Horváth et al. [10] introduce a modified cellular particle filter with Metropolis resampling on the GPU for real-time applications. LibBi [11] is a software package for state-space modelling and Bayesian inference capable of utilizing GPUs. Several particle filters are implemented in LibBi, e.g., particle Markov Chain Monte Carlo (pMCMC) and SMC^2. Other methods in LibBi include the Extended Kalman Filter (EKF) and parameter optimisation routines. Bai and Hu [12] demonstrate particle filter-based data assimilation for simulation of wildfire spread, with parallel sampling and weight computation based on the MapReduce programming model. In a more recent work, Bai et al. [13] describe more efficient routing of particles between processing units in the resampling step of a distributed particle filter.

Other data-assimilation methods have also been subject to GPU-accelerations. Blattner and Yang [14] give a performance study of a GPU-implementation of the local ensemble transform Kalman filter, Wei and Huang [15] explore a GPU-based implementation of the EKF, and Quinn and Abarbanel [16] present a general path integral Monte Carlo approach applied to a neuron model. They all report massive speed-ups on the order 100-1000 over CPU implementations. Theoretical speed-up based on hardware specifications for FLOPS and memory bandwidth is on the order 10 [17].

Assimilation of Lagrangian data is challenging due to the potential complexity of the trajectories and the need for transforming the data into Eularian velocity data (for fixed-grid or spectral numerical models). Apte, Jones and Stuart [18] use particle smoothing for assimilating Lagrangian data from drifters and present three methods for sampling from the exact posterior probability density function based on the Langevin equation and the Metropolis-Hastings algorithm. Their methods are shown to produce better results than the ensemble Kalman filter using perturbed observations. Spiller, Apte and Jones [19] use both particle filtering and smoothing (exact posterior sampling) for assimilating Lagrangian data from gliders and drifters. They propose a new observation operator to deal with the high uncertainty in the locations of the observations. Spiller et al. [20] investigate the divergence of a particle filter for the point-vortex model. They introduce backtracking particle filters and show that the filters outperform EKF for the two-point vortex system. Other methods than particle filters and smoothers have also been successfully implemented [21, 22, 23, 24, 25].

2 The data-assimilation problem

There are many potential sources for errors in the simulation of atmospheric and oceanographic processes. These errors may arise from physical processes missing in the mathematical model, discretization errors in the numerical method, sub-grid effects that can not be resolved in the discretized model, and uncertainties in model parameters, initial conditions, forcing and boundary conditions. Hence, we do not only wish to simulate the behavior of the unknown physical state, denoted by $\boldsymbol{\psi}$, but rather its probability density function (pdf), $p(\boldsymbol{\psi})$. As geophysical applications tend to be very high-dimensional and driven by nonlinear processes, an analytic description of $p(\boldsymbol{\psi})$ is generally unobtainable, and an ensemble-based Monte-Carlo simulation is one way to measure the uncertainties in the system. In its simplest form, ensemble-based statistical simulation consists of a set of N_e independent state vectors $\{\boldsymbol{\psi}_i\}_{i=1,\dots,N_e}$, which are initialized according to uncertainties in the model parameters and initial conditions. The state of each ensemble member is then simulated independently according to the model equation,

$$\boldsymbol{\psi}_i^n = M\left(\boldsymbol{\psi}_i^{n-1}\right) + \boldsymbol{\beta}_i^{n-1}, \quad \text{for } n = 1, 2, \dots, \tag{1}$$

in which the model M evolves the solution deterministically from time t^{n-1} to t^n, and $\boldsymbol{\beta}_i^{n-1}$ is an optional stochastic variable that represents realizations of the errors in the model. The pdf of the system can then be represented through the statistical properties of the resulting ensemble, e.g., as

$$p(\boldsymbol{\psi}^n) = \frac{1}{N_e}\sum_{i=1}^{N_e} \delta\left(\boldsymbol{\psi}^n - \boldsymbol{\psi}_i^n\right), \tag{2}$$

in which δ is the Dirac delta function.

If an observation $\boldsymbol{y}^n$ of the system is available at time t^n, this information can be used to improve the obtained probability density. Typically, the observation is also influenced by uncertainty, as

$$\boldsymbol{y}^n = H\left(\boldsymbol{\psi}^n_{true}\right) + \boldsymbol{\epsilon}^n, \tag{3}$$

in which H is the observation operator that maps the true state $\boldsymbol{\psi}^n_{true}$ to observation space and $\boldsymbol{\epsilon}^n$ is a stochastic observation error. The observations typically only cover parts of the system, so that the size of the observation vector (denoted N_y) is smaller than the size of state vector (denoted N_{ψ}). This is particularly true for geophysical systems, for which it is normal that $N_y \ll N_{\psi}$ (e.g., $\boldsymbol{y}$ can be the value and direction of the ocean current at a single point in space and time). Because of this, we can not simply replace the observed parts of $\boldsymbol{\psi}^n$ with the values in $\boldsymbol{y}^n$ directly, and we have to consider the conditional pdf $p\left(\boldsymbol{\psi}^n|\boldsymbol{y}^n\right)$. The data-assimilation problem consists of finding this conditional density, and its fundamental building block is Bayes theorem:

$$p(\boldsymbol{\psi}^n|\boldsymbol{y}^n) = \frac{p(\boldsymbol{y}^n|\boldsymbol{\psi}^n)p(\boldsymbol{\psi}^n)}{p(\boldsymbol{y}^n)}. \tag{4}$$

The original pdf $p(\boldsymbol{\psi}^n)$ is here termed the *prior probability*, as it represents our understanding of the system prior to assimilating the information in the observation. The *likelihood* $p(\boldsymbol{y}^n|\boldsymbol{\psi}^n)$ expresses the probability of observing $\boldsymbol{y}^n$ under the assumption that $\boldsymbol{\psi}^n$ is the true state of the system. The *marginal probability* $p(\boldsymbol{y}^n)$, i.e., the probability of observing $\boldsymbol{y}^n$, acts mainly as a normalization constant and ensures that the resulting *posterior probability density* is a pdf.

2.1 Standard particle filter

The *standard particle filter* is an ensemble-based data-assimilation technique that uses a direct evaluation of Bayes theorem. Each particle (equivalent to an ensemble member), $\boldsymbol{\psi}_i$, is assigned a weight w_i that gives the relative importance of that particle in the ensemble. Typically, all N_e particles are initialized with weight $w_i^0 = 1/N_e$, as they are sampled independently from the pdf of the initial conditions, $p(\boldsymbol{\psi}^0)$. Each particle is then simulated independently according to (1) until observation time t^n. By applying (4) directly with (2) as the prior density, and by considering the marginal probability as a normalization constant, the posterior distribution is expressed as

$$\begin{aligned} p(\boldsymbol{\psi}^n|\boldsymbol{y}^n) &\propto \sum_{i=1}^{N_e} \frac{p(\boldsymbol{y}^n|\boldsymbol{\psi}_i^n)}{\sum_{j=1}^{N_e} p(\boldsymbol{y}^n|\boldsymbol{\psi}_j^n)} \delta(\boldsymbol{\psi}^n - \boldsymbol{\psi}_i^n) \\ &= \sum_{i=1}^{N_e} w_i^n \delta(\boldsymbol{\psi}^n - \boldsymbol{\psi}_i^n). \end{aligned} \tag{5}$$

Here, the likelihood is used to update the weights w_i^n for each particle, so that the posterior is represented by a weighted discrete distribution. We can evaluate the likelihood if we know the pdf for the observation. For instance, if the observation error is Gaussian, $\epsilon^n \sim N(0, R)$, the weight for particle $\boldsymbol{\psi}_i$ becomes

$$w_i \propto \exp\left[-\frac{1}{2}\left(\boldsymbol{y}^n - H(\boldsymbol{\psi}_i^n)\right)^T R^{-1}\left(\boldsymbol{y}^n - H(\boldsymbol{\psi}_i^n)\right)\right]. \tag{6}$$

As some particles inevitably end up with very low weights, they no longer carry significant statistical value. To improve the statistical coverage in the high-probability regions, the ensemble is *resampled* according to the weight distribution in (6), so that $\{\boldsymbol{\psi}_i^n\}_{i=1,\dots,N_e} \sim p(\boldsymbol{\psi}^n|\boldsymbol{y}^n)$. All weights for the resampled particles are then reset to $1/N_e$. This is known as sequential importance resampling. Several schemes can be used for this resampling [4], and in this work we consider the residual resampling scheme [26]. Note that if the model (1) has $\boldsymbol{\beta} = 0$, it is important that duplicated particles are given a perturbation to avoid ensemble collapse and completely overlapping particle trajectories. With a stochastic model, however, exact duplications will evolve differently through independent realizations of $\boldsymbol{\beta}_i$.

One of the main advantages of the standard particle filter is that it preserves all physical properties throughout the simulation, as the final particles are generated from successful simulation runs and not through manipulation of the state vectors. A drawback, however, is that the ensemble is prone to collapse when the dimension of the observation space increases [3, 4, 5]. In high-dimensional systems, all particles end up in the tail of the likelihood, with the consequence that only very few particles (perhaps even just one) gain a much higher weight than all others. The distribution then collapses as all N_e particles are resampled from few (or a single) particles that have non-zero weights. This problem is often referred to as the *curse of dimensionality*.

2.2 The implicit equal-weights particle filter

One technique used for overcoming the curse of dimensionality is to sample the states $\boldsymbol{\psi}_i^n$ from a *proposal density*, q, with an appropriate compensation in the weights. First, (1) shows that the pdf of the state at time t^n is related to that of the previous time by the Markovian property

$$p(\boldsymbol{\psi}^n) = \int p(\boldsymbol{\psi}^n|\boldsymbol{\psi}^{n-1})p(\boldsymbol{\psi}^{n-1})\, d\boldsymbol{\psi}^{n-1} \approx \frac{1}{N_e}\sum_{i=1}^{N_e} p(\boldsymbol{\psi}^n|\boldsymbol{\psi}_i^{n-1}), \tag{7}$$

where we assumed that all particles have the same weight at time t^{n-1}. In the standard particle filter, we draw the evolution of the particle from $p(\boldsymbol{\psi}^n|\boldsymbol{\psi}_i^{n-1})$, which is equivalent to solving the model equation for one time step. We can choose it differently, by first multiplying and dividing the argument of the integral by a proposal density q and then draw the particle evolution from that density,

$$p(\boldsymbol{\psi}^n) = \frac{1}{N_e}\sum_{i=1}^{N_e} \frac{p(\boldsymbol{\psi}^n|\boldsymbol{\psi}_i^{n-1})}{q_i(\boldsymbol{\psi}^n|\boldsymbol{\psi}_{1:N_e}^{n-1}, y^n)} q_i(\boldsymbol{\psi}^n|\boldsymbol{\psi}_{1:N_e}^{n-1}, y^n). \tag{8}$$

We have large freedom in how to choose q, but the support of q is required to be equal to or larger than the support of $p(\boldsymbol{\psi}^n|\boldsymbol{\psi}_i^{n-1})$, and it should preferably be easy to sample from. Here, the proposal is chosen to be conditioned on the observation $\boldsymbol{y}^n$ and all particle states at the previous time step, $\boldsymbol{\psi}_{1:N_e}^{n-1}$, and it depends on the parent state $\boldsymbol{\psi}_i^{n-1}$ via index i. Using the proposal density in Bayes theorem (4) gives us

$$p(\boldsymbol{\psi}^n|\boldsymbol{y}^n) = \frac{1}{N_e}\sum_{i=1}^{N_e} \frac{p(\boldsymbol{y}^n|\boldsymbol{\psi}^n)p(\boldsymbol{\psi}^n|\boldsymbol{\psi}_i^{n-1})}{p(\boldsymbol{y}^n)q_i(\boldsymbol{\psi}^n|\boldsymbol{\psi}_{1:N_e}^{n-1}, \boldsymbol{y}^n)} q_i(\boldsymbol{\psi}^n|\boldsymbol{\psi}_{1:N_e}^{n-1}, \boldsymbol{y}^n). \tag{9}$$

By now sampling $\boldsymbol{\psi}_i^n \sim q_i(\boldsymbol{\psi}^n|\boldsymbol{\psi}_{1:N_e}^{n-1}, \boldsymbol{y}^n)$, the posterior becomes

$$p(\boldsymbol{\psi}^n|\boldsymbol{y}^n) = \sum_{i=1}^{N_e} w_i^n \delta(\boldsymbol{\psi}^n - \boldsymbol{\psi}_i^n), \quad \text{with} \quad w_i^n = \frac{p(\boldsymbol{y}^n|\boldsymbol{\psi}_i^n)p(\boldsymbol{\psi}_i^n|\boldsymbol{\psi}_i^{n-1})}{N_e p(\boldsymbol{y}^n)q_i(\boldsymbol{\psi}_i^n|\boldsymbol{\psi}_{1:N_e}^{n-1}, \boldsymbol{y}^n)}. \tag{10}$$

One choice of q is the *optimal proposal density* [27], in which $q_i(\boldsymbol{\psi}^n|\boldsymbol{\psi}_{1:N_e}^{n-1}, \boldsymbol{y}^n) = p(\boldsymbol{\psi}_i^n|\boldsymbol{\psi}_i^{n-1}, \boldsymbol{y}^n)$. By considering a linear observation operator H and Gaussian model and observation errors, $\boldsymbol{\beta} \sim N(0, Q)$ and $\boldsymbol{\epsilon} \sim N(0, R)$, the optimal proposal density is equivalent to $N(\boldsymbol{\psi}_i^{n,a}, P)$, with

$$\boldsymbol{\psi}_i^{n,a} = M(\boldsymbol{\psi}_i^{n-1}) + QH^T\left(HQH^T + R\right)^{-1}\boldsymbol{d}_i^n \tag{11}$$

and

$$P = \left(Q^{-1} + H^T R^{-1} H\right)^{-1}, \tag{12}$$

in which

$$\boldsymbol{d}_i^n := \boldsymbol{y}^n - HM(\boldsymbol{\psi}_i^{n-1}) \tag{13}$$

is called the *innovation* for particle i. The proposal is optimal in the sense that it gives optimal variance in the weights for proposals of the form $q(\boldsymbol{\psi}^n|\boldsymbol{\psi}_i^{n-1}, \boldsymbol{y}^n)$, but as it turns out, it is not sufficient to avoid ensemble degeneracy [3, 5, 28].

The main particle filter we will use in this work is an extension of the implicit equal-weights particle filter (IEWPF). In the IEWPF [29], q is chosen similar but not identical to the implicit particle filter [30] by choosing the new particles as

$$\boldsymbol{\psi}_i^n = \boldsymbol{\psi}_i^{n,a} + \alpha_i^{1/2} P^{1/2} \xi_i, \tag{14}$$

in which ξ_i is a draw from the standard multivariate Gaussian distribution $\xi_i \sim N(0, I)$ and α_i is a function of both ξ and ψ_i^{n-1}. Furthermore, we choose α_i such that the weights of all particles become equal to a target weight, which is equal to the lowest optimal proposal weight of all the particles. This choice is needed to ensure that we keep all particles in the ensemble, but comes with two drawbacks. Firstly, when the number of particles increases, the worst particle will be located further and further away from the observations, so the scheme enforces all particles to move further away from the observations. Secondly, numerical experiments show that the spread of the particles becomes underestimated in low-dimensional systems (its behaviour in high-dimensional systems is harder to assess as we do not know the true answer). Not withstanding these negatives, the IEWPF is the first particle filter that has uniform weights in high-dimensional systems.

To alleviate these two issues, Skauvold et al. [31] extended the scheme by proposing an update equation for each particle of the form:

$$\boldsymbol{\psi}_i^n = \boldsymbol{\psi}_i^{n,a} + \alpha_i^{1/2} P^{1/2} \xi_i + \beta^{1/2} P^{1/2} \nu_i, \tag{15}$$

in which ν_i is a second random vector $\nu_i \sim N(0, I)$ and β is a covariance scaling parameter common to all particles. The introduction of the new term enables us to remove the underestimation of the particle spread by tuning β. Furthermore, we can choose α_i and β such that the target weight is equal to the mean of the optimal proposal weights. The consequence of this choice is that the particles are not forced away from the observations when the ensemble size increases. With this, both problems are solved, and this new scheme is the basis for our numerical experiments. Details of the scheme are given in A.

3 Simplified ocean model for massively parallel architectures

Traditional ocean circulation models [2, 32] are generally written to resolve as many of the physical processes in the ocean as possible, and typically consider conservation of mass, momentum, energy, and tracers (salt and temperature) in three dimensions. This makes them very computationally demanding and limits the feasible number of ensemble members. The number of members in an operational ensemble prediction system today is usually between 10 and 100. Instead of a full three-dimensional ocean circulation model, we assume that the vertical velocities are negligible compared to the horizontal movement, and let the nonlinear shallow-water equations in a rotational domain serve as a simplified model. Thus, we vastly reduce the state space of the problem. In operational settings, the simplified model may be initialized based on the most recent ocean state from a traditional ocean circulation model, and be used to forecast short-term ocean currents. Furthermore, drift of Lagrangian objects in the ocean are typically driven by the ocean currents, wind, and wave-induced forces (Stokes drift) [33], whereas in this work we only consider the contribution from the ocean currents.

The shallow-water equations are in the class of hyperbolic conservation laws, which are often solved using explicit finite-volume methods [34]. This class of problems is well-suited for efficient implementation on massively parallel hardware, such as GPUs [35, 36, 37]. By also carefully tailoring the data-assimilation algorithms to use local operations, we are able to run the most computationally demanding parts of the code on the GPU. Control flow and intrinsic serial operations, however, are still carried out on the CPU. This way, we use each processor type for the task which it is best suited for. Through this approach, we can efficiently run an ensemble of a simplified ocean model on commodity-level desktop computers, reducing the requirements for access to supercomputers.

The GPU is an extreme case of a many-core processor, with hundreds or thousands of simple cores. Measured in floating-point operations per second (FLOPS), a standard desktop GPU surpasses the performance of the top supercomputer in the world ten years ago [38], and is today roughly ten times as fast as the CPU. GPUs were initially designed for efficient graphics operations, but have become increasingly popular for general-purpose computing over the last 15 years. Due to their design for optimized throughput of data-parallel operations and low prices driven by the gaming market, they became attractive accelerators when the steadily increasing CPU clock frequency came to an end [39]. Programming languages such as CUDA and OpenCL, and easy access to highly specialized third-party libraries[1], debuggers and profilers, have further contributed to make them accessible for a wide range of computational problems.

The programming model of the GPU is accessed through *kernels*, which are programs written in specialized languages for running on the GPU in a SIMD/SIMT (Single Instruction, Multiple Data/Threads) fashion. The threads are organized in *blocks*, which again are organized in a *grid*. The grid (and blocks) can be one-, two- or three-dimensional, and the ideal choice of block-size configuration, denoted by (b_x, b_y), will vary for different kernels and for different GPUs. Each thread can communicate with other threads in the same block through the *shared memory*, which can be described as a programmable cache or scratchpad memory. Communication between threads in different blocks, however, requires costly global synchronization. The GPU does not share the main CPU memory, and all required data therefore needs to be explicitly transferred between the GPU and CPU. This operation is relatively expensive and should be minimized for optimal performance. For a more thorough introduction to GPU computing; see, e.g., Sanders and Kandrot [40].

To achieve both computational performance and code development efficiency, we treat the computational intensive part of the code and the program flow in different ways. PyCUDA [41] is a Python package that exposes the complete CUDA run-time API and allows us to call native GPU kernels written in CUDA directly from Python. This way, one can write the program flow, as well as pre- and post-processing of the specific applications, in high-level Python, and at the same time ensure that the computationally expensive simulation loop runs as efficient as possible through low-level CUDA C/C++. By taking advantage of widely available and popular packages – including NumPy [42] and matplotlib [43], and environments such as the Jupyter Notebook [44] – the code and experiments can be developed efficiently through rapid prototyping.

In the remainder of this section we give an overview of the model and the model errors, and show how we utilize the GPU to increase computational efficiency.

3.1 The simplified ocean model

The shallow-water equations consider three conserved variables; the elevation η of the free ocean surface relative to its equilibrium level, and the volume transport hu and hv along the abscissa and ordinate, respectively. The equilibrium depth is given by H_{eq} and is here assumed to be constant, so that the full height of the water column becomes $h = H_{eq} + \eta$. With gravitational acceleration g and Coriolis parameter f, the shallow-water equations can be written

$$\begin{aligned}
(\eta)_t + (hu)_x + (hv)_y &= 0, \\
(hu)_t + \left(hu^2 + \frac{1}{2}gh^2\right)_x + (huv)_y &= fhv, \\
(hv)_t + (huv)_x + \left(hv^2 + \frac{1}{2}gh^2\right)_y &= -fhu.
\end{aligned} \tag{16}$$

The equations represent a hyperbolic conservation law, and can be written in vector form as

$$\boldsymbol{\psi}_t + F(\boldsymbol{\psi})_x + G(\boldsymbol{\psi})_y = S_f(\boldsymbol{\psi}), \tag{17}$$

for a state vector $\boldsymbol{\psi} = [\eta, hu, hv]^T$. Here, F and G are flux terms along the absicca and ordinate, respectively, and S_f consists of the source terms due to the Coriolis forces.

The model operator $M(\boldsymbol{\psi})$ will be the numerical scheme that solves (16) and evolves the state forward in time. We use the high-resolution central-upwind scheme proposed by Chertock et al. [45], but with a reformulation that avoids the expensive recursive formulation of Coriolis potential terms [46]. The scheme is designed to be well-balanced with respect to the geostrophic balance,

$$hu = -\frac{gH_{eq}}{f}\frac{\partial \eta}{\partial y} \quad \text{and} \quad hv = \frac{gH_{eq}}{f}\frac{\partial \eta}{\partial x}, \tag{18}$$

[1]BLAS, RNG, FFT, image and signal processing, collective communication primitives, graph analytics, etc.

which permits rotating steady-state solutions by balancing the gravitational and Coriolis forces. The numerical scheme is solved on a Cartesian grid Ω^M consisting of $N_M = n_x \times n_y$ cells. The size of each cell is $\Delta x \times \Delta y$, so that the cell with index (j, k), containing the value $\boldsymbol{\psi}_{j,k}$, is the cell centered at

$$(x_j, y_k) = \left(\left(j + \tfrac{1}{2}\right) \Delta x, \left(k + \tfrac{1}{2}\right) \Delta y\right). \tag{19}$$

The total size of the state vector $\boldsymbol{\psi}$ then becomes $N_{\boldsymbol{\psi}} = 3N_M$. The time integration is solved by a second-order strong-stability-preserving Runge-Kutta method, and the storage requirement for the scheme is therefore $2N_{\boldsymbol{\psi}}$, as the full state must be stored for two consecutive time steps.

The step size of the numerical scheme is limited by the CFL condition,

$$\Delta t_{scheme} \leq \frac{1}{4} \min \left\{ \frac{\Delta x}{\max_{\Omega^M} \left| u \pm \sqrt{g(H_{eq} + \eta)} \right|}, \frac{\Delta y}{\max_{\Omega^M} \left| v \pm \sqrt{g(H_{eq} + \eta)} \right|} \right\}, \tag{20}$$

in which the dominating term is the speed of gravitational waves, $\sqrt{g(H_{eq} + \eta)}$. Even though such waves occur in the ocean, perhaps most notable through tides, their contribution to drifter motion is limited. Eddies and other rotation-driven dynamics are much more important, but they operate on longer timescales. Nevertheless, the CFL-condition in (20) must be satisfied to ensure numerical stability. To run the data-assimilation model on a relevant time scale, we decouple the model operator M from the time step of the numerical scheme, and let the fixed model time step Δt consist of as many Δt_{scheme} steps as necessary. We evaluate the condition in (20) continuously to adapt Δt_{scheme} to the most recent model state, using a Courant number of 0.8.

3.2 Small scale model errors

To account for errors in our model (e.g., missing physics), we introduce small-scale perturbations through the stochastic variable, $\boldsymbol{\beta} = [\delta\eta, \delta hu, \delta hv]^T$, so that $\boldsymbol{\beta}$ is approximately drawn from $N(0, Q)$. This model error is generated by sampling a random vector $\xi \sim N(0, I)$ and applying a covariance operator,

$$\boldsymbol{\beta} = Q^{1/2}\xi. \tag{21}$$

This error is added to the model state after each model time step Δt. We design the covariance operator based on two requirements. First, since we aim to implement all components in the data-assimilation system to run efficiently on massively parallel architectures, we design the covariance operator $Q^{1/2}$ in terms of local operations. Second, it is important that the stochastic model error does not introduce discontinuities or non-physical model states to the solution.

To make the perturbation of the ocean surface $\delta\eta$ sufficiently smooth, it is generated according to a second-order auto-regressive (SOAR) function given by

$$\delta\eta_{j,k} = \sum_{a=1}^{n_x} \sum_{b=1}^{n_y} Q^{1/2}_{SOAR} \left(\Omega_{j,k}, \Omega_{a,b}\right) \xi_{a,b}, \tag{22}$$

in which

$$Q^{1/2}_{SOAR}(\Omega_{j,k}, \Omega_{a,b}) = q_0 \left(1 + \frac{\text{dist}(\Omega_{j,k}, \Omega_{a,b})}{L_0}\right) \exp\left[-\frac{\text{dist}(\Omega_{j,k}, \Omega_{a,b})}{L_0}\right]. \tag{23}$$

Here, q_0 is a scaling parameter for the amplitude of $\delta\eta$, L_0 is a measure of the correlation length scale, and $\text{dist}(\Omega_{j,k}, \Omega_{a,b})$ is the Euclidean distance between the center of the cells with indices (j, k) and (a, b). Since the covariance between points that are far from each other relative to L_0 becomes zero, the computational work can be limited to operate on local data points only, and this satisfies the first design requirement. Equation (22) can then be written as

$$\delta\eta_{j,k} = \sum_{a=j-c_{SOAR}}^{j+c_{SOAR}} \sum_{b=k-c_{SOAR}}^{k+c_{SOAR}} Q^{1/2}_{SOAR} \left(\Omega_{j,k}, \Omega_{a,b}\right) \xi_{a,b}, \tag{24}$$

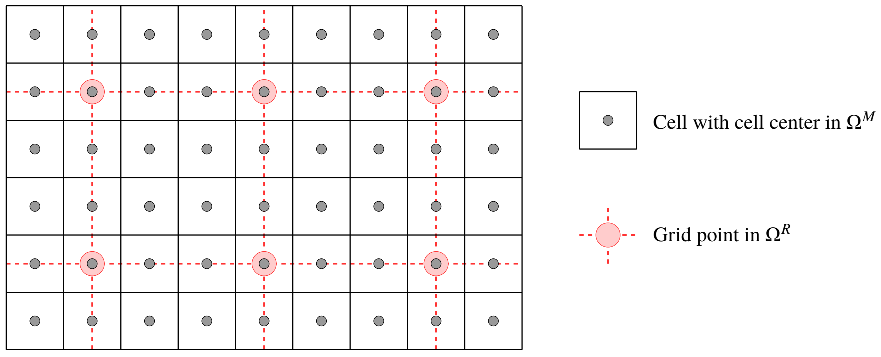


Figure 1: Alignment of nested grids with $c_\Omega = 3$. The grid Ω^M contains cells and is used for evolving the numerical model, whereas the grid Ω^R contains point values and is used for applying the SOAR function on sampled random numbers from $N(0, I)$. For best possible assimilation of observations, an offset can be applied to Ω^R so that one of its grid points is co-located with the cell in Ω^M in which the observation was made.

in which c_{SOAR} is our cut-off value, tuned so that there is no contribution to $\delta\eta_{j,k}$ from a distance larger than $c_{SOAR}\min(\Delta x, \Delta y)$ from cell $\Omega_{j,k}$. Operations such as (24) are very well suited for implementation on the GPU.

A drawback to the expression in (24) is that the computational work and data dependency of the stencil is tightly connected to the ratio between L_0 and the cell size. To have better control of this workload, we introduce a coarse *random number grid* Ω^R, on which the standard normal distributed random numbers ξ are sampled, and apply the SOAR function here. We choose the discretization of Ω^R so that we obtain a good trade-off between computational efficiency of (24), while maintaining a good spread of information within the correlated areas. The coarse grid will have grid cells of size $(\tilde{\Delta x}, \tilde{\Delta y}) = c_\Omega(\Delta x, \Delta y)$, where c_Ω is an odd number representing the coarseness of Ω^R. Values on Ω^R are interpreted as point values, and we denote the number of grid points in Ω^R by N_R. By requiring that c_Ω is odd, we ensure that the point values defined on Ω^R are co-located with cell centers of Ω^M, as show in Figure 1. Furthermore, we choose the coarsening factor c_Ω so that the cut-off factor in (24) can be chosen as $c_{SOAR} = 2$. After having obtained $\delta\eta$ on Ω^R through (24), we use bicubic interpolation, denoted by the operator I_Ω, to obtain cell-averaged values on Ω^M.

To avoid that the perturbation $\boldsymbol{\beta}$ produces non-physical model states (the second design requirement), we use (18) to ensure that $\boldsymbol{\beta}$ is in geostrophic balance. By discretizing (18) with central differences on the Ω^M grid, δhu and δhv are found from $\delta\eta$ by

$$\delta hu_{j,k} = -\frac{gH_{eq}}{f}\frac{\delta\eta_{j,k+1} - \delta\eta_{j,k-1}}{2\Delta y} \quad \text{and} \quad \delta hv_{j,k} = \frac{gH_{eq}}{f}\frac{\delta\eta_{j+1,k} - \delta\eta_{j-1,k}}{2\Delta x}. \tag{25}$$

This operation is denoted by $Q_{GB}^{1/2}$. It should be noted that the derivatives of $\delta\eta$ are approximated by (25), even though they are analytically available directly from the bicubic interpolation. The reason is that geostrophic balance is only maintained by the numerical scheme with respect to the grid resolution. The bicubic surface, however, is continuously defined and will typically contain oscillations on sub-grid scale, meaning that the derivatives of the bicubic surface often will not be represented by the discrete values on the grid. The central differences in (25) are therefore better suited for generating a model state that is in balance under the numerical scheme.

Evaluating the complete model error now consists of four operations,

$$\boldsymbol{\beta} = Q^{1/2}\xi = Q_{GB}^{1/2} I_\Omega Q_{SOAR}^{1/2}\xi, \tag{26}$$

in which the first step is to sample $\xi \sim N(0, I)$. Note that $Q_{GB}^{1/2}$ and $Q_{SOAR}^{1/2}$ are linear operators, whereas I_Ω is a

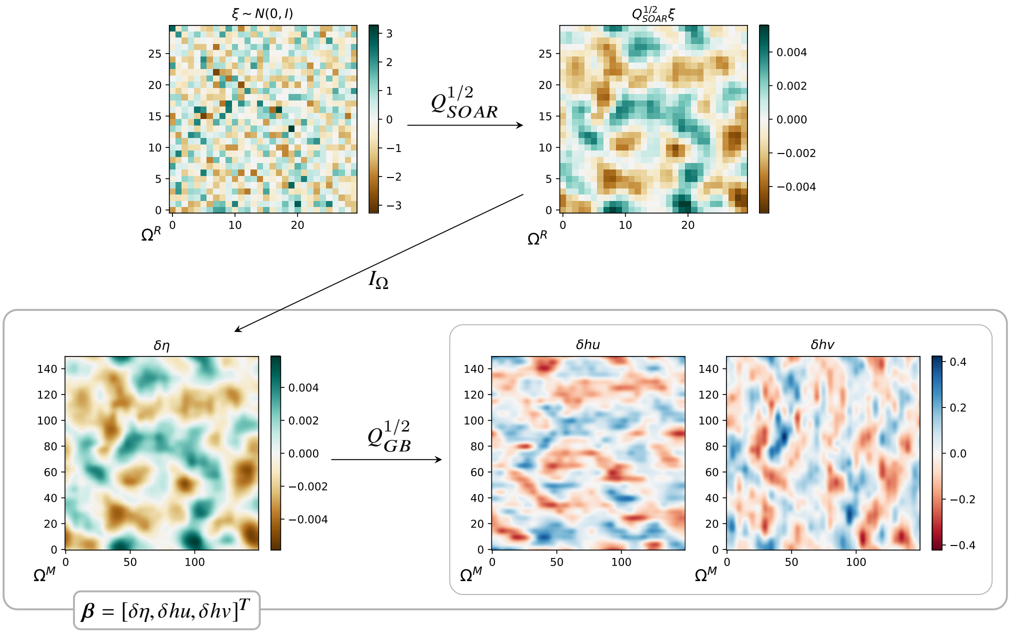


Figure 2: The small scale model perturbation $\boldsymbol{\beta} = [\delta\eta, \delta hu, \delta hv]^T$ is generated by first sampling random numbers from a standard normal distribution $\xi \sim N(0, I)$ on the coarse grid Ω^R. We then give the random field a covariance structure according to the SOAR function $Q^{1/2}_{SOAR}$, before interpolating the coarse random field onto the fine model grid Ω^M through I_Ω to get $\delta\eta$. Finally, we calculate the corresponding momentum δhu and δhv to impose geostrophic balance.

nonlinear stencil. The input and output for each of the operations are

$$\begin{aligned} Q^{1/2}_{SOAR} &: \Omega^R \to \Omega^R, \\ I_\Omega &: \Omega^R \to \Omega^M, \\ Q^{1/2}_{GB} &: \Omega^M \to 3 \times \Omega^M, \end{aligned} \tag{27}$$

making the covariance operator act as

$$Q^{1/2} : \Omega^R \to 3 \times \Omega^M. \tag{28}$$

These operations are illustrated in Figure 2. First, the random field ξ is sampled on the coarse grid Ω^R, and the SOAR operator $Q^{1/2}_{SOAR}$ is applied to generate a coarse correlated field. Then, the correlated field is interpolated onto the computational grid Ω^M using I_Ω, and δhu and δhv are computed to be in geostrophic balance with respect to $\delta\eta$.

It should be noted that our choice of the model error leads to a non-symmetric square root $Q^{1/2}$, and that this implementation-oriented definition of $Q^{1/2}$ makes use of significantly fewer random numbers than variables in the state vector. Using $c_\Omega = 3$, illustrated in Figure 1, as an example, we sample one random number for every nine η variables, and none for hu and hv, since δhu and δhv are computed from (25). This results in one random number for every 27 state variables. To justify why Q is a covariance matrix, we can imagine that all $3 \times \Omega^M$ variables have a corresponding sampled random number, but all those that are not involved in $Q^{1/2}_{SOAR}$ are given very small variance and no correlation to any other variables, so that they become negligible in the above computations. It should finally be noted that because $Q^{1/2}$ is a nonlinear operator due to the bicubic interpolation, the $\boldsymbol{\beta}$'s are not strictly Gaussian distributed. This, however, is not a problem as the covariance of the $\boldsymbol{\beta}$'s is still symmetric positive semi definite.

3.3 Efficient implementation of model errors

The SOAR function, bicubic interpolation, and geostrophic balance are all local stencil operations that are simple to parallelize, as each element of their output can be found independently from all other output elements. Generation of random numbers ξ can further be done through the cuRAND library available through the CUDA toolkit. The sampling of $\boldsymbol{\beta}$ is therefore well-suited for implementation on the GPU.

The SOAR function in (24) with $c_{SOAR} = 2$ consists of a stencil operation depending on 5×5 input values centered on the target cell. We use one GPU thread per output element. To minimize the amount of data read from global memory, all threads within the same block cooperate to read the collectively required input data into shared memory.

In the bicubic interpolation I_Ω, each value in the fine grid Ω^M depends on the 4×4 points in the coarse grid Ω^R that surrounds its position. This means that the $c_\Omega \times c_\Omega$ output values that are located between the same four coarse grid points have overlapping data dependencies. We still apply one GPU thread per output element, and obtain geostrophically balanced δhu and δhv within the same kernel. Each block computes $(b_x + 2) \times (b_y + 2)$ values of $\delta\eta$ and stores them temporarily in shared memory, so that $b_x \times b_y$ values of δhu and δhv efficiently can be computed using (25).

The memory footprint of obtaining $\boldsymbol{\beta}$ is two buffers of size N_R, holding ξ and the result from $Q^{1/2}_{SOAR}\xi$, respectively. The memory footprint of the random number generator comes in addition to this. Note that we never store $\boldsymbol{\beta}$ itself, but add it directly into the state vector $\boldsymbol{\psi}$.

3.4 Synthetic truth and observations

The experiments in this paper are so-called identical twin experiments, meaning that the same model equations are used to generate the synthetic true state and to evolve the ensemble. The true state $\boldsymbol{\psi}_{true}$ is generated from a known set of initial conditions by running the numerical scheme with stochastic model errors as described above. Furthermore, N_D Lagrangian drifters (drifting buoys) are simulated to be advected passively along the ocean current according to a simple forward Euler integration scheme.

Our data-assimilation experiments make use of two types of observations. The first type is based on "GPS" tracking of drifters, and we denote the position of drifter d at observation time t_m by (x^m_d, y^m_d)[2]. Since the location of a drifter at a single point in time gives no information about the underlying ocean currents, we use the difference in two subsequent drifter locations to estimate the drifter velocity, which in our model represents the current. The observation $\boldsymbol{y}^n_d$ then becomes

$$\boldsymbol{y}^m_d = \left[\frac{x^m_d - x^{m-1}_d}{t_m - t_{m-1}} H_{eq}, \frac{y^m_d - y^{m-1}_d}{t_m - t_{m-1}} H_{eq} \right] + \boldsymbol{\epsilon}^m_d, \tag{29}$$

in which $\boldsymbol{\epsilon}^m_d \sim N(0, R)$ is the observation error. Note that the observation is chosen to be an estimate of the state variables hu and hv, but where we have ignored the contribution of the unobserved sea-surface level η. This simplifies the observation operator H to be the state values in the cell corresponding to the drifter position. If drifter d is observed at location (x^m_d, y^m_d), and this is a point within cell $\Omega^M_{j,k}$, the observation operator applied to a particle state $\boldsymbol{\psi}^m_i$ becomes

$$H\left(\boldsymbol{\psi}^m_i, (x^m_d, y^m_d)\right) = \left[(hu^m_i)_{j,k}, (hv^m_i)_{j,k} \right]^T. \tag{30}$$

The size of the observation vector becomes $N_y = 2N_D$.

One challenge with the above observation is the unobserved value of the sea-surface level η, as it in general is not negligible compared to H_{eq}, and therefore introduces a bias in (29). To compensate for this, we use the best available estimate for η, namely the simulated η for each individual particle, and define the innovation related to drifter d for particle i as

$$\boldsymbol{d}^m_{i,d} = \boldsymbol{y}^m_d \frac{H_{eq} + (\eta^m_i)_{j,k}}{H_{eq}} - H\left(\boldsymbol{\psi}^m_i, (x^m_d, y^m_d)\right). \tag{31}$$

The second observation type is observations from moored buoys, referred to simply as moorings in the remainder of the paper, that give Eulerian point measures of the current throughout the entire simulation. To be consistent with

[2]Note that observations might not be available for each model time step, which is the reason for the use of subscript m to distinguish observation time step t^m from from model time step t^n.

(30) and (31), the mooring observations are provided in terms of hu and hv, but ignoring the contribution from η. The observation from mooring μ, located at (x_μ, y_μ) in cell $\Omega^M_{j,k}$, is therefore defined as

$$\boldsymbol{y}^m_\mu = \left[(hu^m_{true})_{j,k} \frac{H_{eq}}{H_{eq} + (\eta^m_{true})_{j,k}}, (hv^m_{true})_{j,k} \frac{H_{eq}}{H_{eq} + (\eta^m_{true})_{j,k}} \right] + \boldsymbol{\epsilon}^m_\mu. \tag{32}$$

As for the drifter observations, the size of the mooring observation vector becomes $N_y = 2N_\mu$, for N_μ moorings.

3.5 Adjoint of the model error operators

Whereas the model error term depends on $Q^{1/2}$ only, the IEWPF algorithm requires that we apply the full Q operator, e.g., in (11). This requires us to express $Q^{1/2,T}$, the adjoint operator for $Q^{1/2} = Q^{1/2}_{GB} I_\Omega Q^{1/2}_{SOAR}$. As mentioned in Section 3.2, $Q^{1/2}$ is not symmetric for our application. The operator $Q^{1/2}_{SOAR}$ is linear and symmetric, however, and therefore its own adjoint $Q^{1/2}_{SOAR} = Q^{1/2,T}_{SOAR}$. The expression for geostrophic balance is close to linear, and $Q^{1/2,T}_{GB}$ is approximated simply by $H_{eq} + \eta \approx H_{eq}$. The bicubic interpolation operator, I_Ω, however, is nonlinear and its adjoint is therefore challenging to express. Our solution to this is to approximate $Q^{1/2,T}$ entirely on the coarse grid Ω^R and define I^T_Ω to be a coarsening operator. The approximate adjoint operator for the model errors is then defined as

$$Q^{1/2,T} \approx Q^{1/2}_{SOAR} Q^{1/2,T}_{GB} I^T_\Omega, \tag{33}$$

with

$$\begin{aligned} I^T_\Omega &: 3 \times \Omega^M \to 3 \times \Omega^R, \\ Q^{1/2,T}_{GB} &: 3 \times \Omega^R \to \Omega^R, \end{aligned} \tag{34}$$

and $Q^{1/2}_{SOAR}$ as in (27), resulting in

$$Q^{1/2,T} : 3 \times \Omega^R \to \Omega^R. \tag{35}$$

The full Q operator is always applied to the adjoint of the observation operator H^T, which maps an observation vector to state space. This means that $Q^{1/2,T}_{GB}$ in all practical sense can be considered to operate on hu and hv at a single grid point only. To preserve the location of these point values, we align the cell containing the observation with a grid point in the coarse grid, by applying an offset on the location of grid points of Ω^R. The observation values can then be mapped directly from their position in Ω^M to the corresponding location in Ω^R. With an observation $\boldsymbol{y} = [y_{hu}, y_{hv}]^T$ located at grid point $\Omega^R_{j,k}$, we apply the adjoint geostrophic balance as

$$\left(Q^{1/2,T}_{GB} I^T_\Omega H^T \boldsymbol{y}\right)_{(l,m)} = \begin{cases} \frac{-g}{H_{eq} f} \frac{1}{2\Delta \tilde{y}} y_{hu} & \text{if } (l,m) = (j, k+1), \\ \frac{g}{H_{eq} f} \frac{1}{2\Delta \tilde{y}} y_{hu} & \text{if } (l,m) = (j, k-1), \\ \frac{g}{H_{eq} f} \frac{1}{2\Delta \tilde{x}} y_{hv} & \text{if } (l,m) = (j+1, k), \\ \frac{-g}{H_{eq} f} \frac{1}{2\Delta \tilde{x}} y_{hv} & \text{if } (l,m) = (j-1, k), \\ 0 & \text{otherwise,} \end{cases} \tag{36}$$

for all grid points $\Omega^R_{l,m} \in \Omega^R$.

4 Efficient implementation of the IEWPF scheme

The objective of this section is to present how IEWPF can be efficiently implemented for a shallow-water model with additive locally defined model errors, using the GPU as a target architecture. The algorithmic nature is not limited to GPUs, and the following approach can be used on other architectures that take advantage of massively parallel operations. Section 2.2 gave a high-level overview of the method, whereas mathematical details important to the implementation are given in A. This description assumes the use of drifter observations.

In this work, we also rely on using single-precision floating point arithmetic. This has a potentially huge impact on performance, as some commodity-level GPUs have single- to double precision ratios of up-to 1:32[3]. Hatfield et al. [47] demonstrate how the accuracy of weather forecasts can be improved through reduced-precision data assimilation. They ran a Lorenz '96 "toy" atmospheric model and the ensemble square root filter at double-, single-, and half-precision, and measured the performance through mean error statistics and rank histograms. By trading reduced precision for increased ensemble size, the authors could reduce the assimilation error and improve forecast accuracy compared to double-precision assimilation.

The starting point of our algorithm is an ensemble of forecast states $\{\boldsymbol{\psi}_i^{n-1}\}_{i=1,\dots,N_e}$ having equal weights at the time step before an observation $\boldsymbol{y}^n$ is available. Each particle is then updated through the following pseudo-code:

1. Obtain the position of the drifter and find the innovations $\boldsymbol{d}_i^n$.
2. Pull each particle towards the observation according to the mean of the optimal proposal density (11). Simultaneously, obtain the value of the ϕ_i, which is a measure of the innovation and defined in (A.6).
3. Sample $\xi_i, \nu_i \sim N(0, I)$, such that $\xi_i \perp \nu_i$, and find the sizes of the two random vectors.
4. Find the parameter β and the target weight w_{target}.
5. Solve the implicit equation given by (A.8) for α_i for each particle.
6. Apply the covariance structures of P to ξ_i and ν_i, and calculate the posterior particle states according to (15).

Figure 3 summarizes the algorithm and shows the relevant equations for each step. The algorithm has only a single synchronization point across all particles at step 4. Further, equations marked in green identify massively data-parallel operations, for which we can execute efficiently on the GPU. The following subsections give details about each of the steps just mentioned.

4.1 Observations and innovations

The innovation $\boldsymbol{d}_i^n$ is a measure of how well the observed currents $\boldsymbol{y}^n$ are represented by each particle state. To obtain this value for IEWPF, each particle is evolved forward in time to the observation time t^n by the model, $\boldsymbol{\psi}_i^{n,f} = M(\boldsymbol{\psi}_i^{n-1})$, but without adding the stochastic model error. The observation also contains the location of each drifter, which is used to look up the relevant parts of the particle state vectors according to (29) and (30).

4.2 Optimal proposal particles

Based on the innovations $\boldsymbol{d}_i^n$, each particle state is pulled towards the observation according to the mean of their individual optimal proposal density, given by (11). For simplicity, we start by considering a case with a single drifter located in cell $\Omega^M_{j,k}$. In this case, the matrix $S = (HQH^T + R)^{-1}$ becomes a 2×2 matrix only and represents a combination of the uncertainty or covariance structure from the model error in observation space and the observation error. Since the correlation pattern that makes up the covariance matrix Q for the model error is the same across the entire domain, HQH^T (and thus also S) becomes independent of the observed drifter position. This means that S can be computed and stored once and for all ahead of the assimilation loop. For now, we assume S is already available, and look at how the particle states are pulled towards the observation. Thereafter, we will get back to how S is pre-computed.

We start by expanding the expression for the mean of the optimal proposal density in (11) by using $Q = Q^{1/2}Q^{1/2,T}$:

$$\begin{aligned}\boldsymbol{\psi}_i^{n,a} &= M(\boldsymbol{\psi}_i^{n-1}) + QH^T(HQH + R)^{-1}\boldsymbol{d}_i^n \\ &= \boldsymbol{\psi}_i^{n,f} + Q_{GB}^{1/2} I_\Omega Q_{SOAR}^{1/2} Q_{SOAR}^{1/2} Q_{GB}^{1/2,T} I_\Omega^T H^T S \boldsymbol{d}_i^n.\end{aligned} \tag{37}$$

[3]Nvidia's GTX series, Maxwell generation GPUs.

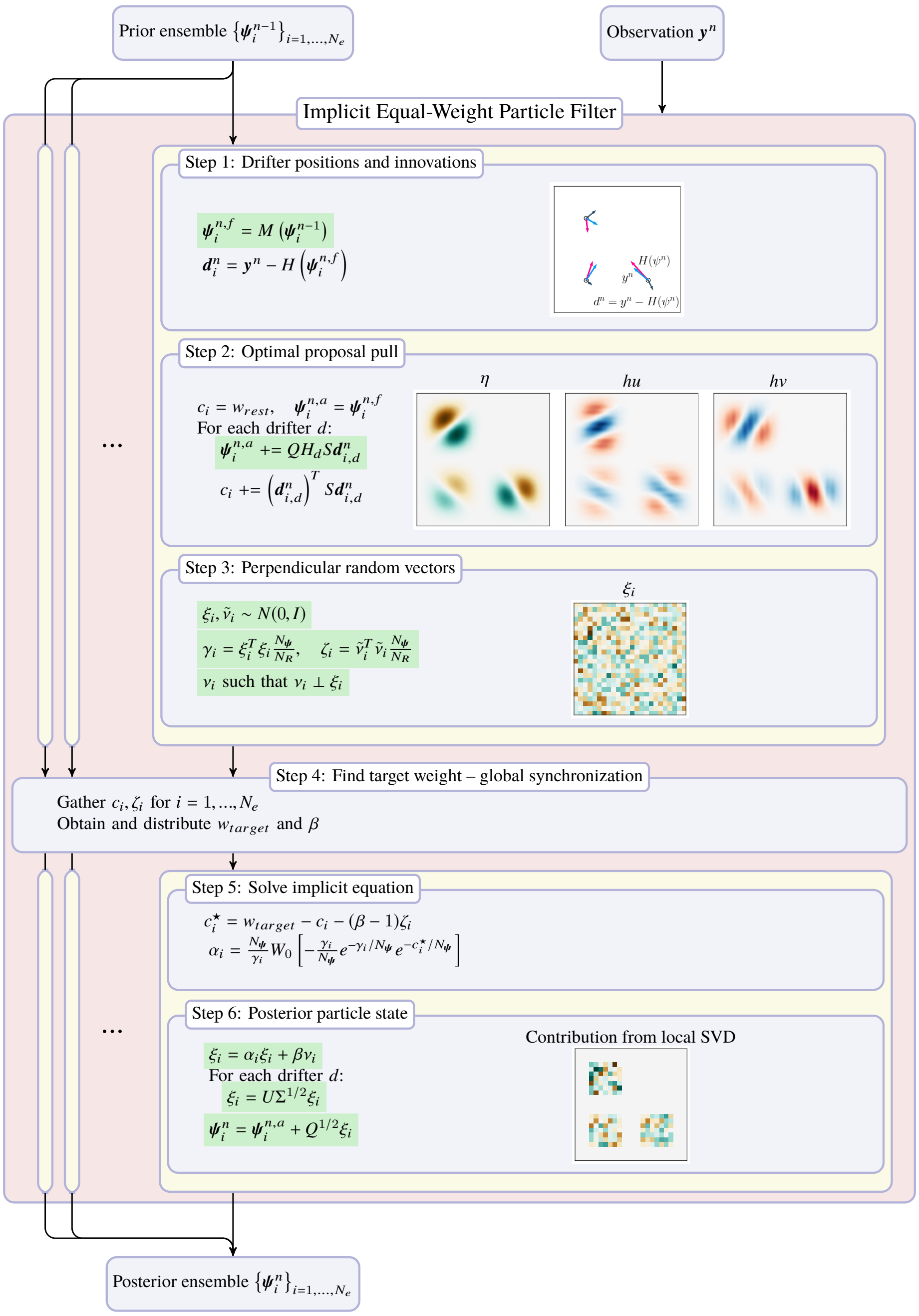

Prior ensemble $\{\psi_i^{n-1}\}_{i=1,\dots,N_e}$
Observation y^n
Implicit Equal-Weight Particle Filter
Step 1: Drifter positions and innovations
$\psi_i^{n,f} = M\left(\psi_i^{n-1}\right)$
$d_i^n = y^n - H\left(\psi_i^{n,f}\right)$
$H(\psi^n)$
y^n
$d^n = y^n - H(\psi^n)$
Step 2: Optimal proposal pull
$c_i = w_{rest}, \quad \psi_i^{n,a} = \psi_i^{n,f}$
For each drifter d:
$\psi_i^{n,a} \mathrel{+}= QH_dSd_{i,d}^n$
$c_i \mathrel{+}= \left(d_{i,d}^n\right)^T Sd_{i,d}^n$
η
hu
hv
...
Step 3: Perpendicular random vectors
$\xi_i, \tilde{\nu}_i \sim N(0, I)$
$\gamma_i = \xi_i^T \xi_i \frac{N_\psi}{N_R}, \quad \zeta_i = \tilde{\nu}_i^T \tilde{\nu}_i \frac{N_\psi}{N_R}$
ν_i such that $\nu_i \perp \xi_i$
ξ_i
Step 4: Find target weight – global synchronization
Gather c_i, ζ_i for $i = 1, \dots, N_e$
Obtain and distribute w_{target} and β
Step 5: Solve implicit equation
$c_i^\star = w_{target} - c_i - (\beta - 1)\zeta_i$
$\alpha_i = \frac{N_\psi}{\gamma_i} W_0 \left[-\frac{\gamma_i}{N_\psi} e^{-\gamma_i/N_\psi} e^{-c_i^\star/N_\psi} \right]$
...
Step 6: Posterior particle state
$\xi_i = \alpha_i \xi_i + \beta \nu_i$
For each drifter d:
$\xi_i = U\Sigma^{1/2}\xi_i$
$\psi_i^n = \psi_i^{n,a} + Q^{1/2}\xi_i$
Contribution from local SVD
Posterior ensemble $\{\psi_i^n\}_{i=1,\dots,N_e}$

Figure 3: An algorithmic overview of one data-assimilation cycle with the implicit equal-weights particle filter. Operations that consists of massively parallel operations are identified on green background and are implemented on a suitable architecture (such as the GPU). During the three first stages, each particle can be handled independently. As stage 4 requires the values of c_i and ζ_i from all particles, this step represents a global synchronization across the entire ensemble. Thereafter, stages 5 and 6 can again be executed independently.

To see how the state of particle i is modified, we go through this expression step-by-step starting from the right. This process is also illustrated in Figure 4.

$\boldsymbol{S d_i^n}$**:** The innovation is the difference between observed and modelled current at the location of the drifter (Figure 4a). This measure is scaled by the combined uncertainty from the observation and the model, represented by S.

$I_\Omega^T H^T S d_i^n$**:** The adjoint observation operator H^T acts on the two-dimensional vector $\boldsymbol{S d_i^n}$ by mapping its two values into state space at the indices representing $hu_{j,k}$ and $hv_{j,k}$. The coarse grid Ω^R is then positioned with an offset so that the center of the cell $\Omega_{j,k}^M$ containing the observation is aligned with a point value in the coarse grid (Figure 4b).

$Q_{GB}^{1/2,T} I_\Omega^T H^T S d_i^n$**:** The adjoint of the geostrophic balance operator spreads the information given by the fields representing the coarse hu and hv onto a single field representing coarse η (Figure 4c), as described by (36).

$Q_{SOAR}^{1/2} Q_{GB}^{1/2,T} I_\Omega^T H^T S d_i^n$**:** The correlation in the surface elevation given by the SOAR function in (24) is applied (Figure 4d), as the final part of the adjoint covariance operator $Q^{1/2,T}$.

$Q_{SOAR}^{1/2} Q_{SOAR}^{1/2} Q_{GB}^{1/2,T} I_\Omega^T H^T S d_i^n$**:** The SOAR function is applied again (Figure 4e), as part of $Q^{1/2}$.

$I_\Omega Q_{SOAR}^{1/2} Q_{SOAR}^{1/2} Q_{GB}^{1/2,T} I_\Omega^T H^T S d_i^n$**:** We interpolate the result from Ω^R to Ω^M, which gives us the final modification applied to η (η in (Figure 4f)).

$Q_{GB}^{1/2} I_\Omega Q_{SOAR}^{1/2} Q_{SOAR}^{1/2} Q_{GB}^{1/2,T} I_\Omega^T H^T S d_i^n$**:** The modifications for hu and hv are found according to the geostrophic balance (Figure 4f) described by (25).

Due to the correlation pattern we have chosen for the model error, all pulls that are added to the particle states are constructed as dipoles to generate a local geostrophically balanced current in a given direction at a given point, without making any other assumptions.

$\boldsymbol{S d_i^n}$ is calculated on the host before it is passed on to a GPU kernel for calculating the adjoint model error operations $Q^{1/2,T}$, and this temporary result (Figure 4d) is written into the buffer originally allocated for normal distributed random numbers, ξ. The remaining operations (applying $Q^{1/2}$ and adding the result to the current particle state) are now identical to imposing the covariance structure of the model error for perturbing the particle state, and this functionality is therefore re-used.

After stepping through the expression in (37), it is also easier to describe how the matrix S is constructed, by first expanding its definition

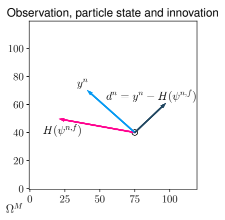
$$S = \left(H Q_{GB}^{1/2} I_\Omega Q_{SOAR}^{1/2} Q_{SOAR}^{1/2} Q_{GB}^{1/2,T} I_\Omega^T H^T + R \right)^{-1}. \tag{38}$$

The observation operator can be considered in matrix form as a $N_\psi \times 2$ matrix consisting of the value 1 in the positions corresponding to the state values $hu_{j,k}$ and $hv_{j,k}$ in the first and second column, respectively, and zeros elsewhere. The process just described in list form and depicted in Figure 4 can therefore be followed by replacing $\boldsymbol{S d_i^n}$ by $[0,1]^T$ and $[1,0]^T$, and applying the observation operator to the final result. This process gives us the two columns of S.

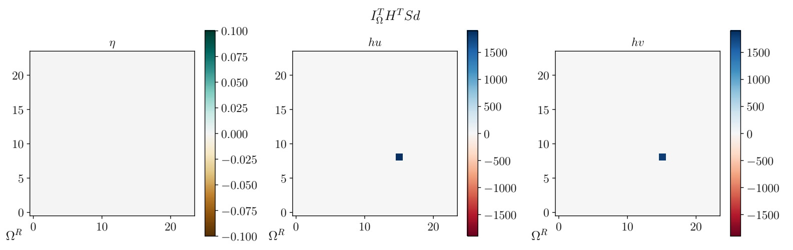
When the observation consists of $N_D > 1$ drifters, we assume that the observations of the drifters are independent of each other, making R diagonal. By also assuming that the drifters are sufficiently far from each other, the resulting matrix from HQH^T becomes block diagonal, which also means that $\left(HQH^T + R\right)^{-1}$ is block diagonal with N_D blocks of the 2×2 matrix S from before. The optimal proposal pull is essentially a local manipulation of the particle state, as seen in Figure 4, and the influence area of this operation is approximately $5\tilde{\Delta x}$ since we have applied the SOAR function twice. In the experiments, however, we do not validate the proximity assumptions for the drifters, and contributions from drifters close to each other are both added to the particle state. This way, the process just described can therefore be applied at each drifter location independently, as seen in step two of Figure 3.

Note that the expression for c_i defined in (A.5) and (A.6) can be calculated almost for free during this step. As each drifter is handled independently, the contributions from all the drifters are summed as

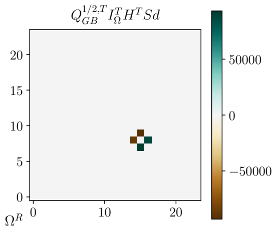
$$c_i = -\log\left(w_i^{n-1}\right) + \sum_{d=1}^{N_D} \boldsymbol{d}_{i,d}^{n,T} \boldsymbol{S d}_{i,d}^n. \tag{39}$$


Observation, particle state and innovation
y^n
$d^n = y^n - H(\psi^{n,f})$
$H(\psi^{n,f})$
Ω^M

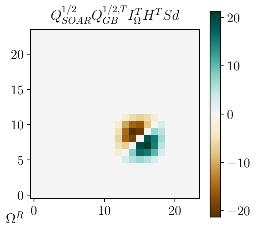
(a) Observation $\boldsymbol{y}^n$, observed forecasted particle state $H(\boldsymbol{\psi}^{n,f})$, and the innovation $\boldsymbol{d}^n$.



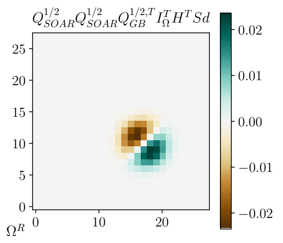
(b) Innovation $\boldsymbol{d}^n$ scaled with the model and observation uncertainty S in the coarse grid state space Ω^R. The coarse grid has been centered onto the observation position. Note that $\eta = 0$ as the observation contains no information on η.



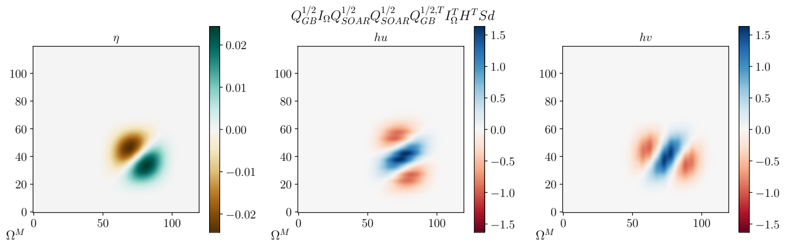
(c) $Q_{GB}^{1/2,T}$ takes the values from hu and hv over to neighbouring grid points in η.



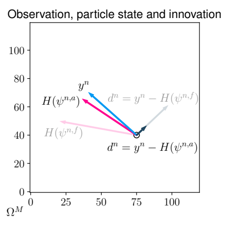
(d) Correlation applied through $Q_{SOAR}^{1/2}$.



(e) Correlation applied through $Q_{SOAR}^{1/2}$ again.



(f) The resulting pull in η is obtained by interpolating the previous result with I_Ω, and applying $Q_{GB}^{1/2}$ to get the pull for hu and hv.


Observation, particle state and innovation
y^n
$H(\psi^{n,a})$
$d^n = y^n - H(\psi^{n,f})$
$H(\psi^{n,f})$
$d^n = y^n - H(\psi^{n,a})$
Ω^M

(g) By looking at the optimal proposal state in observation space, we see that the new particle state is much more similar to the observation than before.

Figure 4: The process of constructing the pull used to obtain the optimal proposal particle state, required in the second step of the IEWPF algorithm. Note that the shown example is exaggerated for illustrative purposes.

The most computationally efficient way to calculate the optimal proposal pull would include adding the contributions from all drifters to the coarse grid before applying the second SOAR function and the interpolation, meaning that $Q^{1/2}$ would only have to be applied once. However, it is essential that the optimal proposal pull is applied at the drifter position with the precision of the computational grid Ω^M, which means that the offset to align the drifter cell to the coarse grid point might be different for each drifter. A coloring scheme could be constructed to maximize parallel processing of the drifters, but this performance optimization has not been realized in our implementation.

4.3 Sampling perpendicular random vectors

The next step is to sample $\xi_i, \nu_i \sim N(0, I)$ in such a way that they become perpendicular. This is achieved by first sampling $\xi_i, \tilde{\nu}_i \sim N(0, I)$ independently. We then decompose $\tilde{\nu}_i = \tilde{\nu}_{i,\parallel} + \tilde{\nu}_{i,\perp}$, so that $\tilde{\nu}_{i,\parallel}$ and $\tilde{\nu}_{i,\perp}$ become parallel and perpendicular to ξ_i, respectively, meaning that

$$\tilde{\nu}_{i,\perp} = \tilde{\nu}_i - \tilde{\nu}_{i,\parallel} = \tilde{\nu}_i - \frac{\tilde{\nu}_i^T \xi_i}{\xi_i^T \xi_i} \xi_i. \tag{40}$$

We then scale $\tilde{\nu}_{i,\perp}$ to have the same length as $\tilde{\nu}_i$, and get

$$\nu_i = \sqrt{\frac{\tilde{\nu}_i^T \tilde{\nu}_i}{\tilde{\nu}_{i,\perp}^T \tilde{\nu}_{i,\perp}}} \tilde{\nu}_{i,\perp}. \tag{41}$$

By using (40) for $\tilde{\nu}_{i,\perp}$ in (41), ν_i can be expressed as

$$\nu_i = \sqrt{\frac{\tilde{\nu}_i^T \tilde{\nu}_i}{\tilde{\nu}_i^T \tilde{\nu}_i - a_i \tilde{\nu}_i^T \xi_i}} \left(\tilde{\nu}_i - a_i \xi_i\right), \qquad a_i = \frac{\tilde{\nu}_i^T \xi_i}{\xi_i^T \xi_i}. \tag{42}$$

This shows that we need to compute the three dot products $\xi_i^T \xi_i$, $\tilde{\nu}_i^T \tilde{\nu}_i$ and $\tilde{\nu}_i^T \xi_i$. Since these dot products have overlapping data dependencies, they can be efficiently found within a single kernel using a common tree-based reduction approach [48]. Finally, $\tilde{\nu}_i$ can be transformed to ν_i element-wise and in-place. Note that this process resembles the Gram-Schmidt orthogonalization process, with preserved vector sizes.

During these computations, we store the values for $\xi_i^T \xi_i$ and $\nu_i^T \nu_i = \tilde{\nu}_i^T \tilde{\nu}_i$, as they are needed for the parameters γ_i and ζ_i, respectively, for solving the implicit equation in step 5. However, as discussed in Section 3.2, the normal distributed random numbers in ξ_i and ν_i do not represent the entire state vector. The derivation of the IEWPF algorithm from Section 2.2 and A assume that $\xi_i^T \xi, \nu_i^T \nu_i \approx N_\psi \pm \sqrt{2N_\psi}$. Since our $\nu_i, \xi_i \in \mathbb{R}^{N_R}$, this assumption is not satisfied directly. To remedy this, we apply an appropriate scaling to the two dot products, and use

$$\gamma_i = \xi_i^T \xi_i \frac{N_\psi}{N_R}, \qquad \zeta_i = \nu_i^T \nu_i \frac{N_\psi}{N_R}. \tag{43}$$

4.4 Target weight and β

To calculate the target weight w_{target} and β, we need to obtain c_i, γ_i and ζ_i for all particles $i = 1, 2, ..., N_e$ in the ensemble. This step represents a global synchronization point in the algorithm. Once all three parameters are provided by all particles, we can calculate w_{target} and β from (A.7) and (A.16), respectively.

4.5 Solving the implicit equation

The final two stages of the algorithm are again independent for all particles. First, $c_i^\star$ is found according to (A.14) and constitutes the final piece for the implicit equation for α_i, given by (A.11). As described in A, the solution for α_i is obtained by using the Lambert W function, which is a scalar operation for each particle.

4.6 Posterior particle states

The final step of IEWPF is to perturb the particles so that they all obtain the target weight, giving the ensemble the correct posterior variance. We need to apply the covariance structure of P to the random fields ν_i and ξ_i, meaning that we seek an expression for $P^{1/2}$ in terms of (preferably) local operations. Instead of using P on the form given in (12), it can be written as

$$\begin{aligned} P =& Q - QH^T \left(HQH^T + R\right)^{-1} HQ \\ =& Q^{1/2} \left(I - Q^{1/2,T} H^T \left(HQH^T + R\right)^{-1} HQ^{1/2}\right) Q^{1/2,T}. \end{aligned} \tag{44}$$

Since there is no easy way to express the operator square root of the expression in the parenthesis in (44), we consider the operations as matrices and seek its singular value decomposition (SVD) by constructing matrices U and V and a diagonal matrix Σ so that

$$\begin{aligned} U\Sigma V^H &= I - Q^{1/2,T} H^T \left(HQH^T + R \right)^{-1} HQ^{1/2} \\ &= I - Q^{1/2}_{SOAR} Q^{1/2,T}_{GB} I^T_\Omega H^T SHQ^{1/2}_{GB} I_\Omega Q^{1/2}_{SOAR}. \end{aligned} \tag{45}$$

This allows us to apply the covariance structure P to a sample $\xi_i \sim N(0, I)$ by

$$P^{1/2}\xi_i = Q^{1/2} U \Sigma^{1/2} \xi_i. \tag{46}$$

For the computation of the SVD, we consider the case in which $\Omega^M = \Omega^R$, meaning that the interpolation and coarsening operators are simplified to the identity. Note that this approximation makes the matrix $Q^{1/2}$ linear, so all operations are well defined. Starting from the right in the parenthesis expression in (44), we step through the operations interpreted as matrices and investigate the structure of non-zero values. This process is illustrated for a small domain consisting of 10×10 cells in Figure 5.

$Q^{1/2}_{SOAR}$: Symmetric matrix of size $N_M \times N_M$, describing the covariance structure defined by the SOAR function in (23) and (24). By using $c_{SOAR} = 2$, the value in each grid cell is given a correlation with a grid cell block of size 5×5 centered on itself. This means that each row of $Q^{1/2}_{SOAR}$ has 25 non-zero values (Figure 5a).

$Q^{1/2}_{GB} Q^{1/2}_{SOAR}$: A $3N_M \times N_M$ matrix, in which the first N_M rows are equal to $Q^{1/2}_{SOAR}$. The middle and lower N_M rows are the results from applying a central difference formula on values of $Q^{1/2}_{SOAR}$ in the y- and x-direction, respectively. These rows have 35 non-zero values on column indices representing 7×5 and 5×7 grid blocks for the middle and lower matrix block, respectively (Figure 5b).

$HQ^{1/2}_{GB} Q^{1/2}_{SOAR}$: The observation operation extracts values of the rows representing $hu_{j,k}$ and $hv_{j,k}$ only, giving us a $2 \times N_M$ matrix with 35 non-zero values for each row (Figure 5c).

$SHQ^{1/2}_{GB} Q^{1/2}_{SOAR}$: All values are scaled by the matrix S representing model and observation uncertainty. The non-zero pattern is not affect by this operation (Figure 5c).

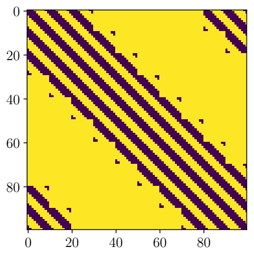
$H^T SHQ^{1/2}_{GB} Q^{1/2}_{SOAR}$: The two rows are mapped back into state space, and inserted into an otherwise zero matrix of size $3N_M \times N_M$ at the rows with indices representing $hu_{j,k}$ and $hv_{j,k}$ (Figure 5d).

$Q^{1/2,T}_{GB} H^T SHQ^{1/2}_{GB} Q^{1/2}_{SOAR}$: The adjoint of the geostrophic balance operator maps the rows representing volume transport to the η-field based on adjoint central differences, resulting in an $N_M \times N_M$ matrix. This means that the row representing $hu_{j,k}$ has non-zero values in rows representing cells $\Omega_{j,k-1}$ and $\Omega_{j,k+1}$, and similarly the row representing $hv_{j,k}$ has non-zero data in the row representing $\Omega_{j-1,k}$ and $\Omega_{j+1,k}$. There are now four rows with 35 non-zero values each (Figure 5e).

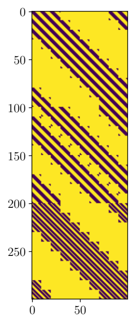
$Q^{1/2}_{SOAR} Q^{1/2,T}_{GB} H^T SHQ^{1/2}_{GB} Q^{1/2}_{SOAR}$: Finally, we apply the SOAR function and each of the existing four non-zero rows are mapped to 25 rows representing a 5×5 grid cell block in the resulting matrix. Considering the overlap between these blocks, we get an $N_M \times N_M$ matrix with 45 non-zero rows, each containing 45 non-zero values. The rows represent a 7×7 grid cell block centered in cell $\Omega_{j,k}$ with a single cell missing in each of the four corners.

$I - Q^{1/2}_{SOAR} Q^{1/2,T}_{GB} H^T SHQ^{1/2}_{GB} Q^{1/2}_{SOAR}$: The final matrix is a $N_M \times N_M$ matrix equal to the identity except for the 45 rows representing the 7×7 grid cell block centered in the cell in which the observation was made.

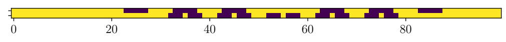
Due to the structure of the problem, the computations for finding the SVD can be greatly simplified by ignoring all rows that are equal to the identity. To further simplify the structure of the code, we include the corners and consider the



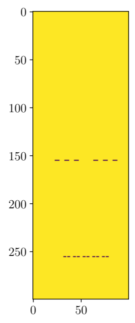
(a) $Q_{SOAR}^{1/2}$



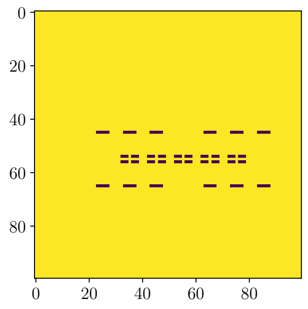
(b) $Q_{GB}^{1/2}Q_{SOAR}^{1/2}$



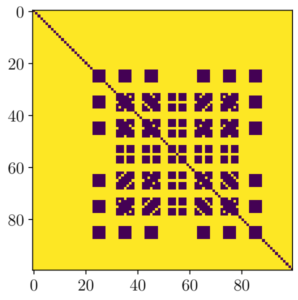
(c) $SHQ_{GB}^{1/2}Q_{SOAR}^{1/2}$



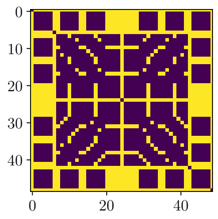
(d) $H^T SHQ_{GB}^{1/2}Q_{SOAR}^{1/2}$



(e) $Q_{GB}^{1/2,T} H^T SHQ_{GB}^{1/2}Q_{SOAR}^{1/2}$



(f) $I - Q_{SOAR}^{1/2}Q_{GB}^{1/2,T} H^T SHQ_{GB}^{1/2}Q_{SOAR}^{1/2}$



(g) The local $7^2 \times 7^2$ block required to represent $I - Q_{SOAR}^{1/2}Q_{GB}^{1/2,T} H^T SHQ_{GB}^{1/2}Q_{SOAR}^{1/2}$

Figure 5: The non-zero patterns that emerge when computing the parenthesis expression for the covariance P in (44) on a small domain consisting of 10×10 cells. (a) The covariance operators $Q_{SOAR}^{1/2}$ and $Q_{GB}^{1/2}$ are interpreted as matrices, meaning that $Q_{SOAR}^{1/2}$ becomes a 100×100 matrix. (b) After applying $Q_{GB}^{1/2}$ we get an extra 200 rows, representing hu and hv in addition to η in every cell. (c) We extract the rows corresponding to the observation and scale them by S, before (d) the values are mapped back to state space. (e) We then apply $Q_{GB}^{1/2,T}$, before (f) applying $Q_{SOAR}^{1/2}$ and subtracting the result from the identity. (g) We ignore the part that is equal to the identity and are left with a covariance matrix describing the 7×7 cell block centered on the observation. Note that by increasing the domain to 100×100 cells, the matrix representing $Q_{SOAR}^{1/2}$ will become $10\,000 \times 10\,000$, but the dense local block (g) will still remain the same.

complete 7×7 grid cell block. This results in a 49×49 matrix described by the above process (Figure 5g), and we can obtain the SVD from this much smaller matrix instead of from the full covariance matrix. When applying $P^{1/2}$ to ξ_i we can then apply the obtained $U\Sigma^{1/2}$ locally according to the observed location of the drifter, before applying $Q^{1/2}$ to values defined in the entire domain as before. In fact, if we assume spatially constant equilibrium depth H and Coriolis parameter f, the structure of the 49×49 non-identity block becomes the same for all drifter positions. This enables us to pre-compute the local SVD matrix ahead of the data-assimilation loop and apply it directly for each observation. Short-term prediction of drift trajectories is in general a local problem, and a constant Coriolis parameter can therefore even for real-world cases be a valid assumption. A constant depth, however, is a more restrictive assumption, and if this is not satisfied, the SVD needs to be computed at each observation. Furthermore, the method outlined here requires that the locations of the observations are further away from land or a non-periodic boundary than the extent of the correlation radius.

Now, if we consider a case with $\Omega^M \neq \Omega^R$, we would need to handle two significant issues. First, the interpolation would result in a much larger local non-zero structure, and thus a larger local matrix $U\Sigma^{1/2}$, requiring more storage, and becoming more expensive to apply. Second, since ξ_i and ν_i are defined for all coarse grid points, we need to use the same offset for co-locating points across the entire domain. Since the drifters are most likely to be located in cells that require different offsets, we will not be able to apply the SVD structure accurately on top of all drifters. Because of these two reasons, we have chosen to define $U\Sigma^{1/2}$ on the coarse grid only, also when $\Omega^M \neq \Omega^R$, which enables us to apply the 49×49 pre-computed matrix $U\Sigma^{1/2}$ to the random field. The $Q^{1/2}$ operator then spreads this information to all three conserved variables on the computational grid. Since this term structure is applied to values that are sampled randomly (ξ_i and ν_i), the simplification does not introduce significant errors.

Finally, we note that $Q^{1/2}$ and $U\Sigma^{1/2}$ are linear operations. Instead of applying the covariance structure first to ξ_i and then to ν_i, we add the scaled random fields before applying $P^{1/2}$. The final posterior particle states in (15) are then obtained by

$$\psi_i^n = \psi_i^{n,a} + P^{1/2}\left(\beta^{1/2}\nu_i + \alpha_i^{1/2}\xi_i\right). \tag{47}$$

5 Experimental results

Here, we describe an experimental setup for experimenting with the data-assimilation method discussed in the previous sections. First, we produce rank histograms to show that the IEWPF method produces statistically sound forecasts and thereby is a valid method for data assimilation. We continue with drift trajectory forecasts through a series of experiments using different numbers of observations from drifters and moorings. This is followed by an illustration of how the standard particle filter collapses for the same case, even when starting from an ensemble that is centered around the true state with low variance. Finally, we measure the computational performance to determine the workload in the IEWPF compared to simply advancing the model.

To get a synthetic but near-realistic ocean model state to represent ψ_{true}, we take inspiration from a test case for validating shallow-water models on a rotating sphere suggested by Galewsky et al. [49]. This test case describes a steady-state solution in which an eastward atmospheric jet is balanced by a smoothed step function in the thickness of the fluid layer due to the sphere's rotation. When a small perturbation is introduced in η, the jet develops instabilities after some time and produces a state that is dominated by complex currents and eddies. By running a simulation from the steady-state but adding random model errors, the case has a chaotic behaviour in which instabilities develop in different places and in different ways for independent simulation runs. This enables us to produce a challenging test case with realistic features for data-assimilation experiments.

We transform the Galewsky test case to a flat two-dimensional rectangular domain with a constant Coriolis parameter. To make the case even more interesting, a second jet is introduced in the opposite direction south of the original jet, so that the balancing ocean surface ends up at an equivalent level at the northern and southern boundary, allowing us to use periodic boundary conditions at all four boundaries. We introduce parameters so that the case represents oceanic flow, rather than atmospheric, and model the domain after the Barents Sea, using a rectangular domain that covers 1110 km $\times$ 666 km, divided into 500×300 cells with $\Delta x = \Delta y = 2220$ m. Further, $g = 9.806$ m/s, $f = 1.405 \cdot 10^{-4}$ s^{-1} (corresponding to 75 degrees north), and a constant equilibrium depth $H_{eq} = 230$ m. Cross sections of the initial steady state for η and hu are shown in Figure 6, and the initial condition for hv is zero. The model time step is chosen as $\Delta t = 60$ s, and the time step in the numerical scheme Δt_{scheme} is dynamically adjusted

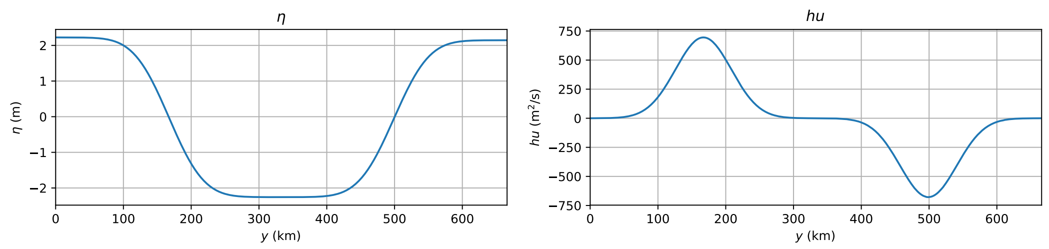


Figure 6: The cross-section along the y-axis of the steady-state initial conditions for the unstable double jet case.

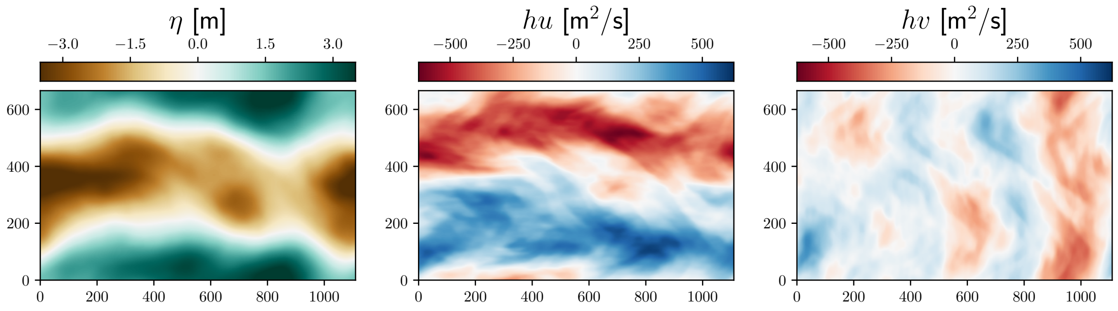


Figure 7: A possible model state after 10 days, resulting from running the shallow-water simulation with additive model errors from the steady-state shown in Figure 6. From left to right, the figures show the surface elevation η, and the volume transport hu and hv in x- and y-direction, respectively. All x- and y-axes are given in km. The realized model state displayed here is also used as the true state for the drift forecast experiments in Section 5.2.

according to the CFL-condition in (20) with a Courant number of 0.8. We use a model error amplitude $q_0 = 2.5 \cdot 10^{-4}$, coarsening factor $c_\Omega = 5$, and model error length scale $L_0 = \frac{3}{4}\tilde{\Delta x}$. Figure 7 shows an example of a model state produced by these parameters after ten simulation days. This is also the true state that is used in the drift trajectory forecasting experiments in Section 5.2.

In all following experiments we consider observations from drifters and moorings, according to the description in Section 3.4. The observation error consists of a measurement error and a representation error and is rarely trivial to quantify in geophysical systems. The measurement error is related to the precision of the instruments that are used to make the observation, e.g., the precision of the GPS used for drifter experiments, and is typically given by the instruction handbook for the relevant instrument. The representation error, on the other hand, should reflect how well the observed measure represents the simulated variables, e.g., how well the mean current along the drifter trajectory represents the cell-averaged depth-integrated water transport. As we here use an identical twin experiment, we can consider the representation error to be small, and we assume that the instruments involved have good accuracy. All experiments hence use $R = I$.

The true state for our experiments is pre-generated by a single thirteen day simulation, during which drifters and moorings are added to the model at the start of day three, and imperfect observations from them are written to file every five minutes. We have used 64 drifters and 240 moorings which are initiated throughout the domain in an 8×8 and a 12×20 pattern, respectively. The experiments use different subsets of these observations for data assimilation. The experimental setup is then divided into three phases (see Figure 8):

Day 0–2: Spin-up period to let the ensemble members start to develop independent instabilities in the two jets. We only perform the spin-up of the ensemble once, so that all experiments start from the same initial ensemble at

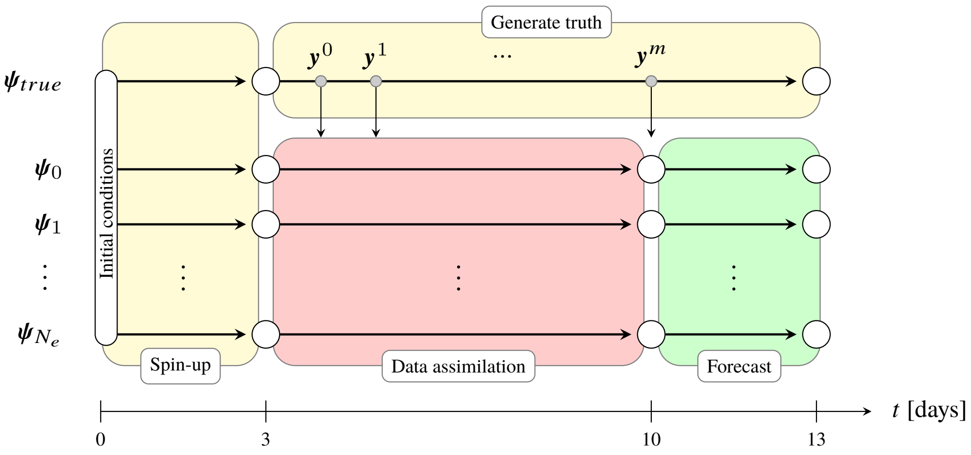


Figure 8: Overview of the drift trajectory ensemble forecast experiments. Before the experiments start, an ensemble of size N_e is spun up from a common initial state, and regular observations from the synthetic truth is generated (yellow). Experiments start at day three, with a seven day period of data assimilation, during which the observations are used to guide the ensemble towards the true state (red). At day 10, there are no more observations, and the ensemble runs a drift trajectory forecast with the latest observed drifter positions as starting point (green).

the beginning of day three.

Day 3–9: Observations from the pre-generated truth are assimilated into the ensemble.

Day 10–12: This is the forecasting period. Drifters are added to all ensemble members at the observed drifter positions at the start of day ten. Each ensemble member runs independently to generate three-day drift trajectory forecasts for all drifters.

We use an ensemble size of $N_e = 100$ where not stated otherwise, as this size will typically fit on a single desktop with a commodity-level GPU.

5.1 Rank histograms for IEWPF

Prior to using the IEWPF scheme for forecasting experiments, we aim to evaluate the quality of the method. One way to do so is by creating a so-called rank histogram by running a large number of independent data-assimilation experiments, and for each of them find the *rank* of a chosen observed variable within the ensemble. For instance, choosing an observed variable $hu_{j,k}$, the ensemble members are sorted based on the value of $(hu_i)_{j,k} + \epsilon_i$, in which $\epsilon_i \sim N(0, R)$ represents the observation error, from lowest to highest. The rank is then the position that the observed value for $hu_{j,k}$ takes when inserted into this sorted list. A histogram is then generated from the number of appearances for each of the $N_e + 1$ possible ranks over all the forecasting experiments. Since a rank can be considered to be a sample from the inverse cumulative distribution of the state density function, the rank histogram is expected to be flat when the data-assimilation method works as intended.

For simplicity of discussion, assume for a moment that there is no observation error and that the posterior distribution of $(hu_i)_{j,k}$ is Gaussian. This corresponds to a high concentration of ensemble members with values close to some mean value, and gradually fewer ensemble members further away from this mean. The mean value is our best estimate for $(hu_{true})_{j,k}$, whereas the spread in the ensemble represents the uncertainty of this estimate. Most likely, the truth should be close to the mean, but since the density of $(hu_i)_{j,k}$ is high around the mean, the resulting rank is sensitive to small variations of $(hu_{true})_{j,k}$. On the other hand, there is a chance of finding the truth in the tail of the ensemble as well. The probability for a tail value to be exactly equal to the truth is very low, but since the density of values in the

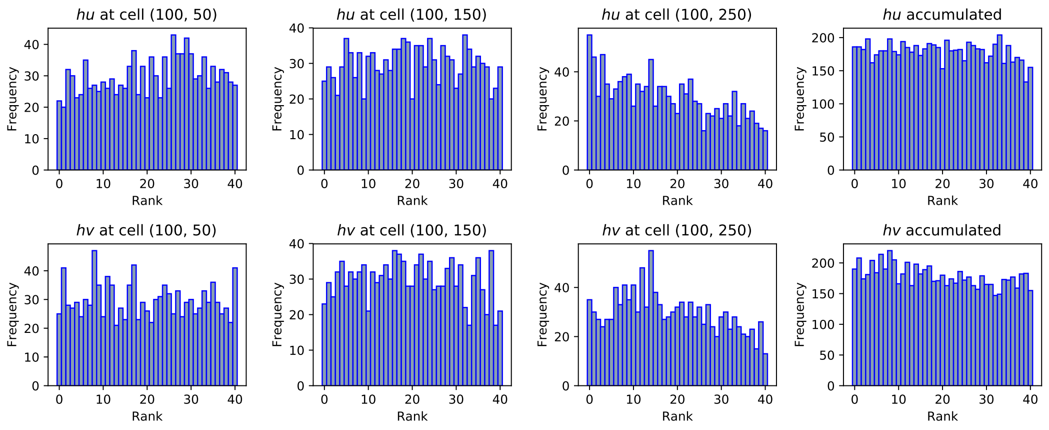


Figure 9: Rank histograms based on 1222 experiments of one hour ensemble forecasts. The rank histograms are generated for hu and hv at cells $(100, y)$ for $y = \{0, 50, 100, ..., 250\}$. The accumulated rank histograms are the sum of the rank at all these cells, and they are considered independent of each other. Most of the generated rank histograms resembles uniform distributions, such as for cells $(100, 50)$ and $(100, 150)$, whereas some are more irregular, such as for cell $(100, 250)$. The accumulated rank histograms are flat.

tail also is low, the truth can take a larger range of values and still obtain the same rank. Ideally, the probability should be the same for obtaining any rank for any forecast experiment, meaning that the rank histogram created from a large number of such experiments should resemble a uniform distribution. If the ensemble constantly fails to represent the uncertainty of the observed state, the histogram takes other forms. For instance, if all ensemble members are too close to the mean, the ranks corresponding to ensemble values at the tail of the ensemble will be over-represented, creating a U-shaped rank histogram. This indicates that the ensemble is under-dispersed. On the other hand, a hill-shaped rank histogram means that the spread in the ensemble forecast is too large to represent the true values, meaning that the ensemble is over-dispersed. This discussion holds for any posterior distribution, not only Gaussian. For a more detailed discussion on the interpretation of rank histograms, see Hamill [50].

We generate rank histograms from 1222 data-assimilation experiments using $N_e = 40$ ensemble members and observations from independently generated truths every five minutes from the 120 moorings in the western half of the domain. The ensemble is initialized by a random sample from the hundred spun up initial conditions generated for the forecast experiments and is run with data assimilation for six hours. We find the ranks from a one hour ensemble forecast from this state at simulation time three days and seven hours. In this time range, we assume that variables located 50 cells apart from each other can be considered as independent and have therefore generated rank histograms for hu and hv at cells $(100, y)$ for $y = \{0, 50, 100, ..., 250\}$. Additionally, since these values are assumed to be independent, an accumulated rank histogram consisting of the sum of all these ranks is created as well. Figure 9 displays a selection of the generated rank histograms. Most of them resemble uniform distributions, such as those shown for cells $(100, 50)$ and $(100, 150)$, but there are also some that display a more irregular trend, such as the one for cell $(100, 250)$. The accumulated rank histograms shown in the rightmost column of the figure also resemble a uniform distribution. In total, these results indicate that our implementation of the IEWPF gives ensemble forecasts with good statistical quality.

5.2 Drift trajectory forecasting experiments

We now turn to drift trajectory forecasting experiments and compare how well the ensemble manages to represent the truth using different sets of observations. We start by investigating how well the ensemble is able to capture the true state at day ten (see Figure 7), by looking at the ensemble means in Figure 10, and ensemble variances in Figure 11.

We then proceed to look at the drift trajectory forecasts for two selected drifters, shown in Figures 13–15, in which the dark lines illustrate the drift trajectory obtained in the generated truth. All Figures 10–15 are organized so that each row corresponds to a single experiment. In Figures 10, 11, and 12, the columns represent the ensemble mean, standard deviation, and root square error, respectively, of state variables η, hu and hv, from left to right, whereas in Figures 13 - 15 each column shows forecasts for a given time range.

With 64 drifters available, it is infeasible to show and discuss forecast results for all of them, and we have therefore chosen to illustrate the results using two different drifters. Drifter number 24 represents a drifter that is located in an area still dominated by one of the jets, and its true drift trajectory follows a smooth path with high velocity. For this drifter, we look at both the long-term and short-term ensemble forecasts in Figures 13 and 14, respectively. In the end of this experiment section, we show that this drifter is representative for the majority of drifters. Drifter number 2, on the other hand, represents a particularly difficult drifter to forecast, as it was at rest at the start of the forecast, before changing direction. Its long-term forecasts are shown in Figure 15.

5.2.1 Experiment A: No data assimilation

The first experiment is a pure Monte Carlo forecasting experiment, in which all ensemble members run independently without any knowledge of the truth. Without any observations to guide the ensemble, the entire space of possible model states that can be reached from the instable initial conditions are explored, and the ensemble mean at day ten, shown in the top row in Figure 10, illustrates the chaotic nature of the chosen test case. Random perturbations of the state vector lead to very different developments of cross-jet momentum at different locations in the domain. Even though individual realizations contain clear cross-jet currents (see Figure 7), the ensemble mean is almost completely smooth, dominated by two opposing jets in the x-direction and almost no action for hv, and resembles a smoothed version of the initial conditions. The ensemble standard deviation, shown in the top row of Figure 11, is more or less the same all over the domain and we see the structure of the true state in the error at the top row of Figure 12.

In the top row of Figure 13, we see that the ensemble forecast suggests that drifter 24 is heading eastward. The ensemble mean trajectory accurately describes the true trajectory, but the variance in the ensemble is very large, indicating high uncertainty in the contribution of north/south currents. As time goes on, the forecast diverges to cover a large portion of the entire domain, as shown in the three day forecast in Figure 14. For drifter 2 the one day forecast in Figure 15 suggests that there are possible drift trajectories in directions, resulting in an ensemble mean that is statically located at the drifter's initial position. As the forecasted trajectories hit the dominant jets, the long-term forecast covers the diagonal from the northwest to the southeast corner of the domain.

5.2.2 Experiment B: Assimilating data from ten drifters

We now observe the locations of ten drifters, and assimilate the underlying currents based on their movements. The ten drifters are hand-picked to cover as large portion of the domain as possible, and both drifter 2 and drifter 24 are included. The second row of Figure 10 shows the ensemble mean at day ten, which is more similar to the mean obtained using no data assimilation than the true state itself (see Figure 7). There are however some localized patches of features in the mean for hu, corresponding to the last observed drifter positions. By looking at the model state standard deviation and error in the second rows of Figures 11 and 12, respectively, the latest drifter positions can be clearly identified by the areas of very low standard deviation and error in hu and hv. Note, however, that the largest standard deviation is also found close to the drifters. As we assimilate an observed current by adding a correcting local dipole at the drifter locations, we might introduce a larger error close to the drifter as a side effect, where opposite currents are needed for maintaining the geostrophic balance. It is also possible to spot traces of the drifters' movements from the patterns in the standard deviation plots, as some of the drifters have tails of low or high standard deviation.

In the short-term forecast in the second row of Figure 13, we see great improvement in the six hour forecast over the assimilation-free forecast, as almost the entire ensemble of drift trajectories starts moving straight eastwards with lower spread. The forecast after 12 hours is improved as well, but when turning to the long-term forecast in Figure 14, it becomes harder to see any significant differences in the forecast quality. The same applies to the forecast for drifter 2 in Figure 15, again with a non-moving ensemble mean trajectory.

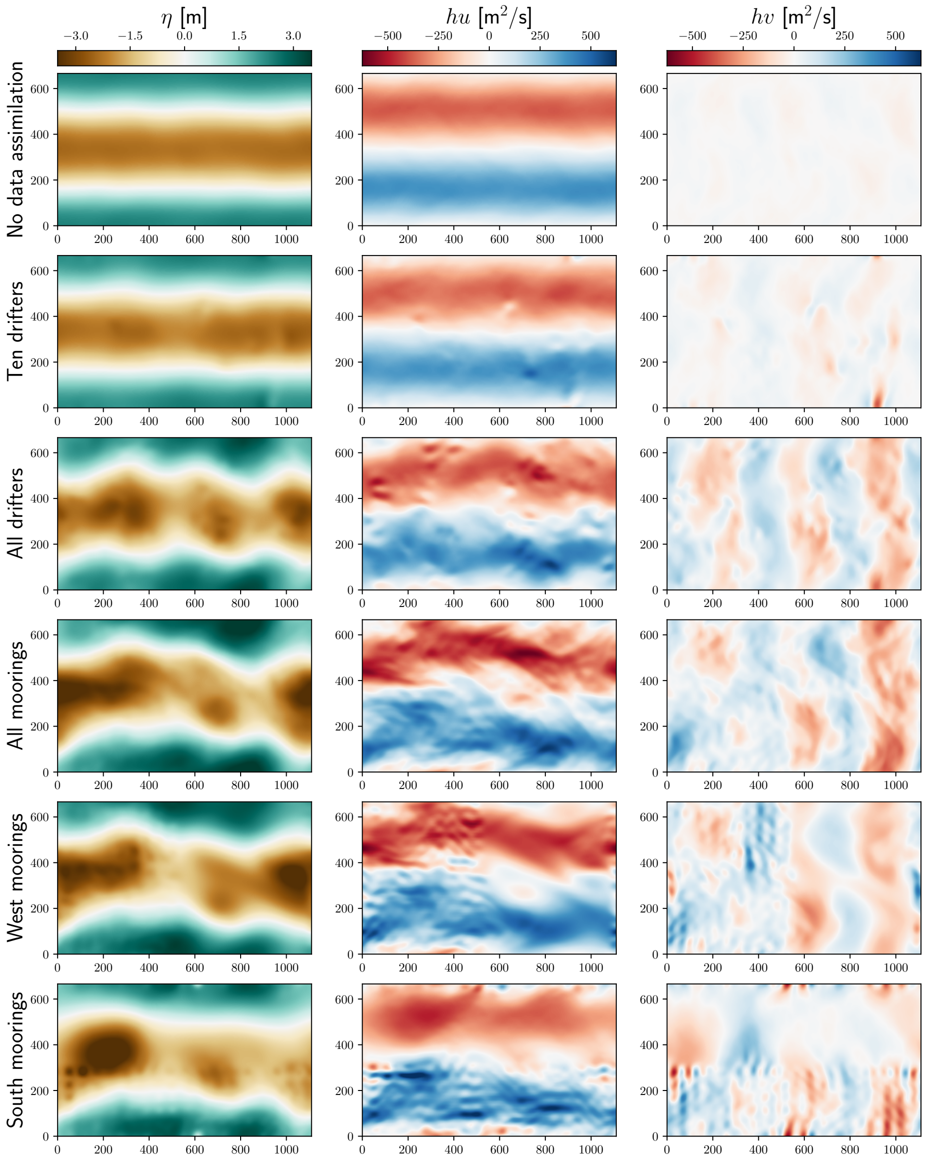


Figure 10: Ensemble means for the state variables sea-surface level (η), jet flow (hu), and cross-jet flow (hv) at simulation day ten, when the data-assimilation period ends and the forecasting starts. The rows represent the six forecast experiments. The x- and y-axes are in km, and all figures cover the entire computational domain. The top row illustrates the chaotic nature of the test case, as the experiment without using data-assimilation results in a steady-state ensemble mean. Through observations from drifters, some localized details are captured, as seen in rows two and three. The ensemble mean from the experiment using all mooring observations in row four gives a very good representation of the true state. The final two rows show the impact of flow-dependent information transport, as only half of the domain is observed in these experiments.

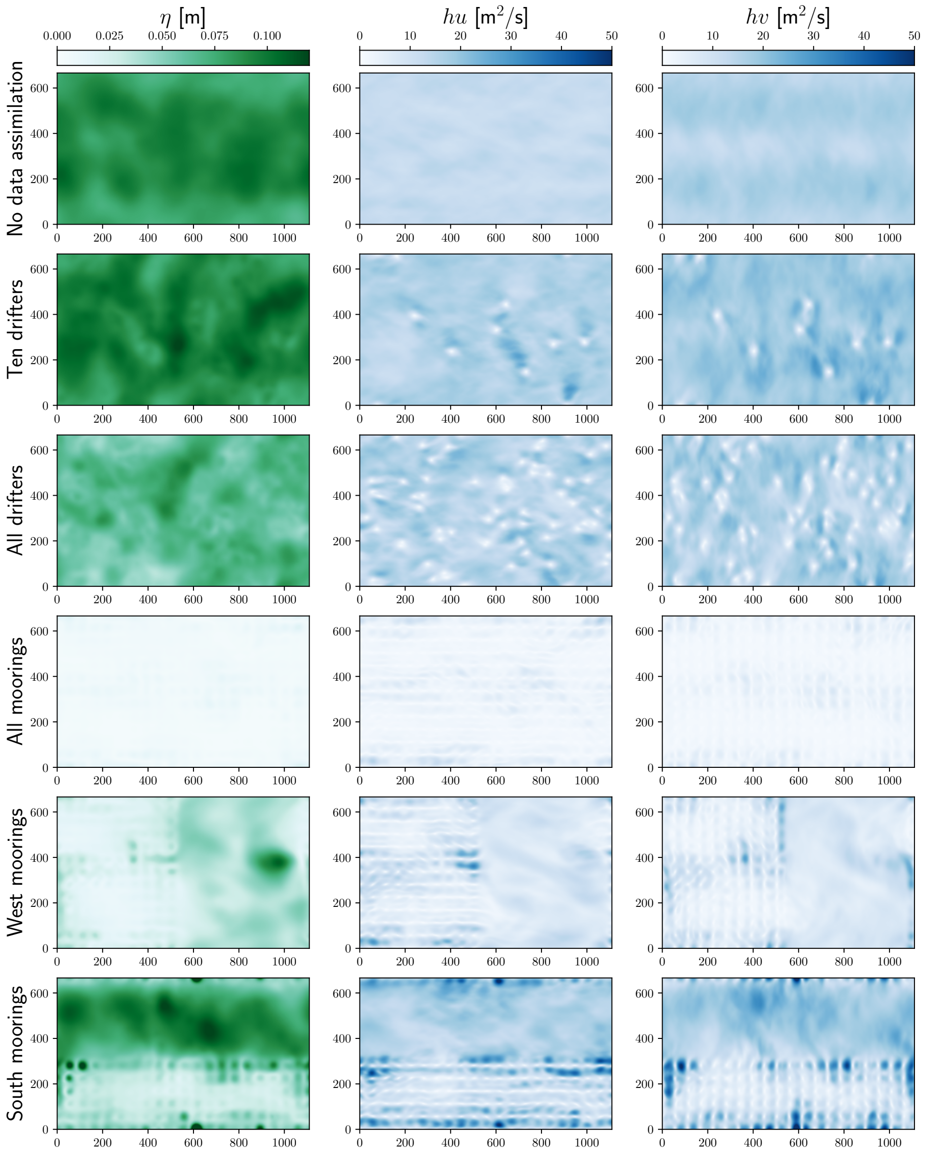


Figure 11: Ensemble standard deviation for state variables sea-surface level (η), jet flow (hu), and cross-jet flow (hv) at simulation day ten, when the data-assimilation period ends and the forecasting starts. The rows represent the six forecast experiments. All x- and y-axes are in km, and cover the entire computational domain. The top row shows almost equal standard deviation throughout the domain when no data assimilation is applied. In rows two and three, the assimilated drifter observations can be clearly seen as local areas with low standard deviation. Row four uses observations from all moorings, resulting in very low standard deviation throughout the domain. The standard deviation increases between the moorings when only half of the them are observed, as seen in rows five and six. We also see that there is a large benefit in observing parts of both jets, contrary to observing one jet fully, as the standard deviation is lower in the fifth row than in the sixth.

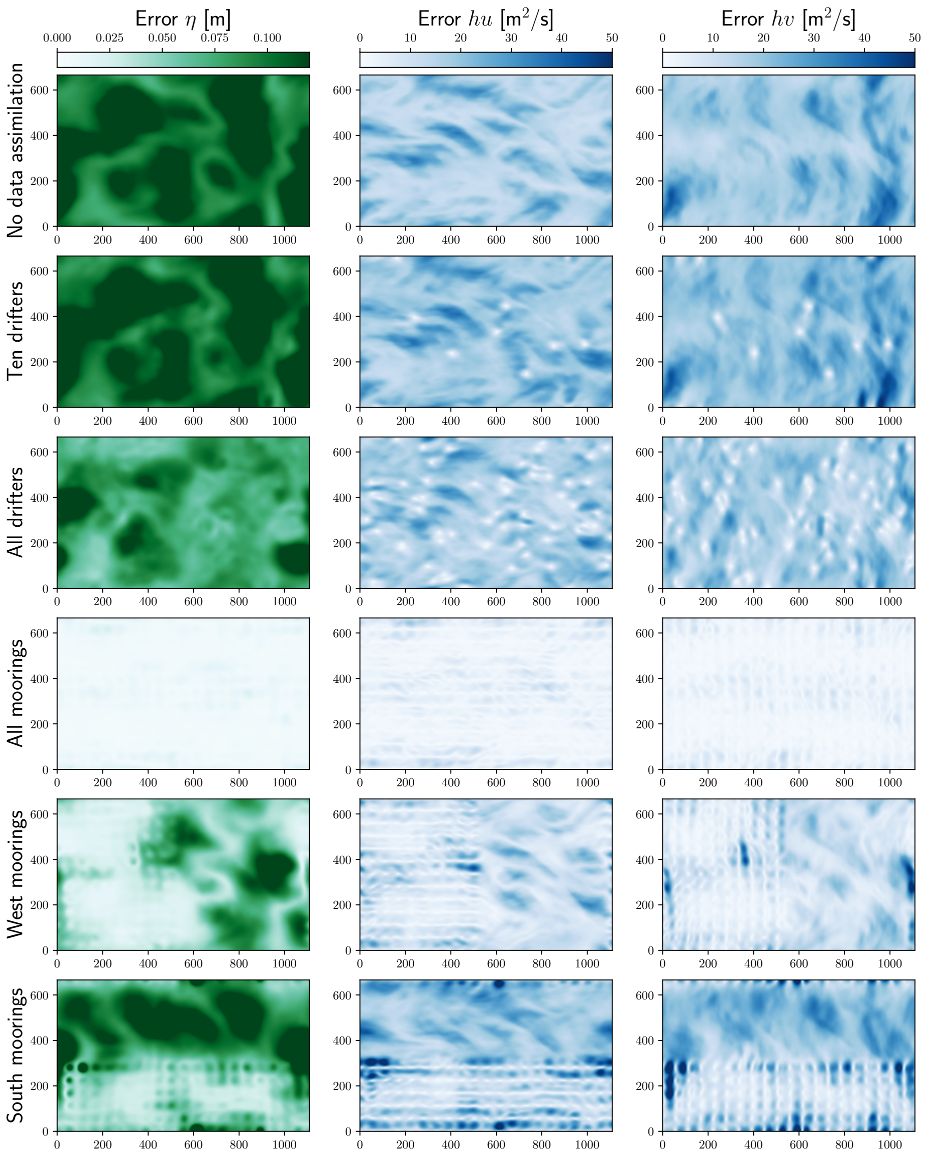


Figure 12: Ensemble root square error for state variables sea-surface level (η), jet flow (hu), and cross-jet flow (hv) at simulation day ten, when the data-assimilation period ends and the forecasting starts. The rows represent the six forecast experiments. All x- and y-axes are in km, and cover the entire computational domain. The top two rows clearly show the structure of the flow of the true state reflected in the error, and the second and third row show the final observed drifter positions in the same way as for the standard deviation. When observing all moorings in the fourth row, the error is small throughout the domain. The final two rows show clearly that the error is smaller when observing the western half of the domain, contrary to the southern half.

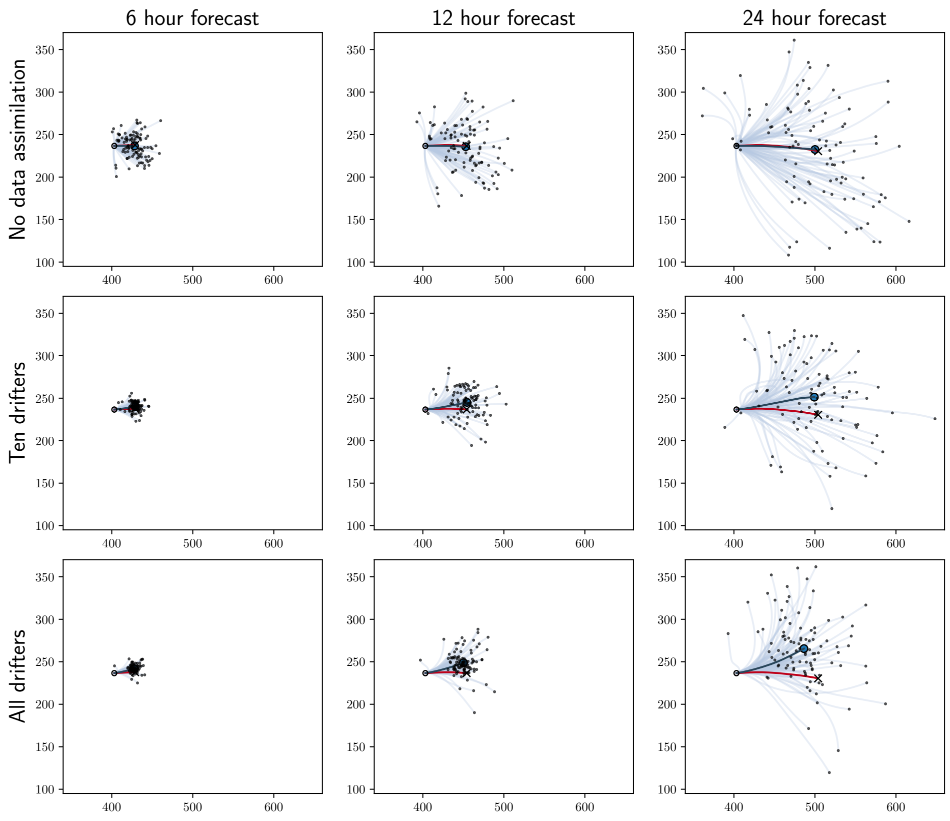


Figure 13: Short-range ensemble drift trajectory forecasts after six, twelve, and 24 hours for drifter 24, using no data assimilation (top row), observations from ten drifters (middle row), and observations from all 64 drifters (bottom row). Trajectories from each ensemble member is shown as a light blue line ending in a small black circle, whereas the dark blue lines represent the ensemble mean. The red line ending in an x is the true drift trajectory. The values along the x- and y-axes are given in km, and only the relevant part of the domain is considered. The forecast at six hours is greatly improved by using observations from ten drifters, but the advantage is almost lost after 24 hours. Further improvements are made using observations from all 64 drifters. Even though the ensemble mean is perfectly on top of the true trajectory in the experiment without data assimilation, the spread is very large.

5.2.3 Experiment C: Assimilating data from all 64 drifters

By using observations from all 64 drifters, we see a large change in the ensemble mean at day ten, presented in the third row of Figure 10. The border between the mean eastward and westward jets in hu is no longer a straight line, and there are more features seen for hv. Even though some of the features resemble the truth, such as the shape of the main parts of the eastward current and the location of the north and south bands in hv, there are other features that are less correct, e.g., the continuity of the north and south bands in hv. As in experiment B, the drifter locations can be seen from the standard deviation of hu and hv, in the third row of Figure 11, and we also note that there are larger areas between the drifters with lower standard deviation than before. In the third row of Figure 12, we see that the overall error is lower than for the previous experiment, meaning that by using observations from an increased number of drifters we are able to improve the model state in a larger portion of the domain.

By comparing Figures 11 and 12, it can be noticed that the values in the error can be higher than those of the standard deviation locally. This does not come as a surprise as the error represents a random variable related to the

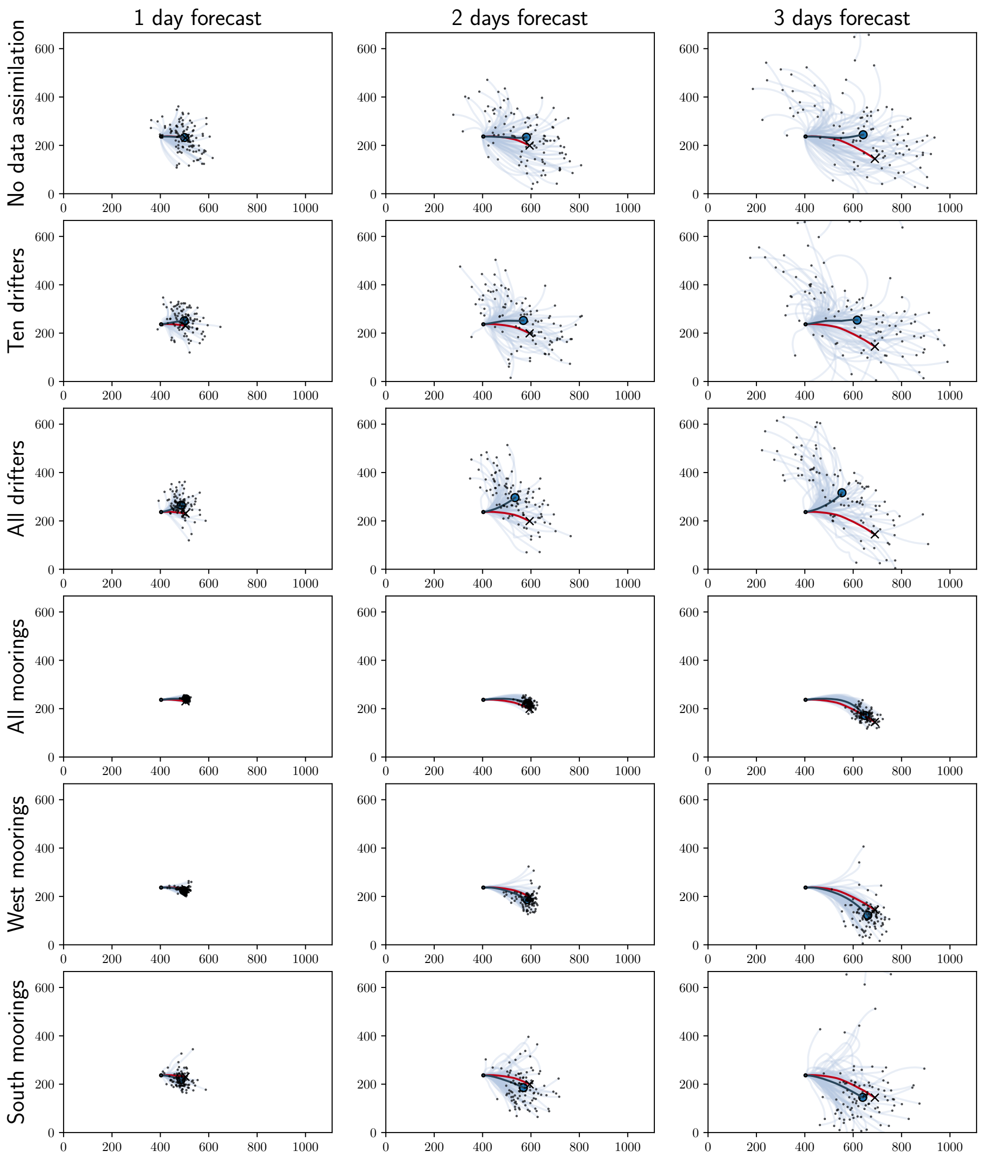


Figure 14: Long-range drift trajectory forecasts after one, two and three days for drifter 24. Each row corresponds to a different set of observations. Each figure shows the entire computational domain, with values in km on both axes. Forecast trajectories from each ensemble member is shown as a light blue line ending with a small black circle, the dark blue lines represent the ensemble means, and the red lines ending in x are the true drift trajectory. In this time range, observations obtained from the drifters are of limited value, as the top three rows are qualitatively similar. The use of mooring observations in the fourth row, however, makes the forecast very accurate even for as long as three days. The two last rows show experiments using mooring observations from half the domain, and illustrates the benefit by partly observing both jets (west moorings), compared to fully observing one jet (south moorings).

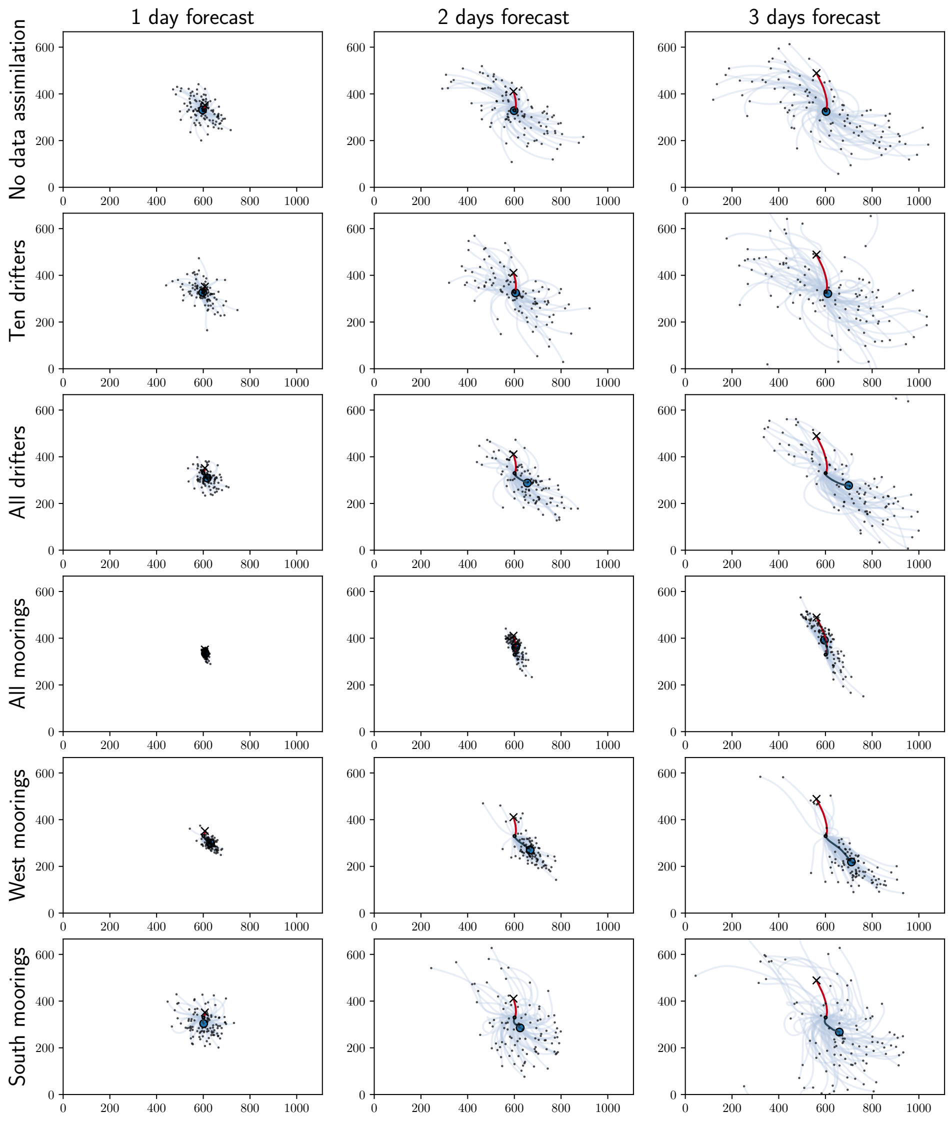


Figure 15: Long-term drift trajectory forecast after one, two and three days for drifter 2. This drifter is particularly hard to forecast, as it is at a complete stop while changing direction at the start of the forecast. Each figure shows the entire computational domain, with values in km on both axes. Forecasted trajectories from all ensemble members are shown in light blue lines ending with a small black circle, the dark blue lines represent the ensemble means, and the red lines ending in x are the true drift trajectory. The use of drifter observations give a limited improvement in the forecast on these time ranges, and the ensemble means in the first two experiment are static at the drifter's initial positions. Observations of all moorings give a large impact on the forecast quality, as seen in the fourth row, and the forecast shows a 75% chance of the drifter drifting northwards. The experiments using observations from half the domain only, give forecasts that show higher probability for southward drift, but there are still a few ensemble members allowing for northwards drift in both cases.

true state, whereas the standard deviation is a statistical moment. The domain averages of each field in Figures 11 and 12 are quite similar, however, suggesting that the data assimilation is doing a good job. This can also be confirmed by the rank histograms in Figure 9, and will be discussed further in relation to the time evolution of the drifter forecasts in Section 5.2.7.

The short-term forecasts for drifter 24 in the third row of Figure 13 have slightly lower spread compared to using ten drifters only, as could be expected. At six hours, the forecast is quite confident in the location of the drifter, and there is less uncertainty in the twelve and 24 hour forecasts as well. By looking at the long-term forecast in Figure 14, however, the quality drops and is again comparable to the previous two experiments. However, we see that the ensemble mean trajectory is no longer static, showing that the majority of ensemble members move south-east.

5.2.4 Experiment D: Assimilating data from all 240 moorings

In this experiment, we assimilate observations from all the 240 moorings that are placed equidistantly throughout the domain. The distances between the moorings are 55 km, corresponding to 25 grid cells. Even though we observe only approximately 0.1% of the state variable, the observations are dense enough for the covariance structures from two neighbouring observations to be overlapping, meaning that the observational coverage is quite good. This is also seen in the ensemble mean after ten days in the fourth row of Figure 10, which is almost indistinguishable from the true state shown in Figure 7. Both the standard deviation and the error are very low throughout the domain, as seen in the forth rows of Figures 11 and 12, and even the unobserved variable η is correctly captured by the ensemble. Note again that both these quantities are of the same size, which indicates that the particle filter works as intended.

The trajectory for drifter 24 shown in Figure 14 is very good, with a very confident forecast and accurate ensemble mean trajectory even at day three. The trajectory of all ensemble members show the same general characteristics by a steady eastern flow with a southward bend, disagreeing only slightly on the strength of these currents. The forecast for the challenging drifter 2 in Figure 15 is also much more accurate than the previous three experiments, but the forecast has a higher spread than for drifter 24 at day three. Most ensemble members stay close to the initial position during the first day. The forecast is then divided, with approximately 75% of the drifters moving northwest, and the last quarter moving southeast. We see that the true trajectory is found among the most likely outcome, to the north. Note that we do not show the short-term forecast trajectories for this and the following two experiments, since the long-term forecasts in Figures 14 and 15 show sufficient information to discuss their results.

5.2.5 Experiment E: Assimilating data from moorings in only the western half of the domain

The last two experiments explore how well the ensemble mean is able to represent the true state if observations come from only half of the domain. We start by using the 120 moorings in the western half only, resulting in the ensemble mean, standard deviation, and error shown in the fifth rows of Figures 10, 11 and 12, respectively. The first thing to notice is that the ensemble mean appears to be less smooth than the true state in the observed area. These slightly noisy features are most dominant in the southwest and northeast corners of the observed area, corresponding to where unobserved water enters the observed part of the domain. The reason for this can be that the signal flowing into the observed area likely need a stronger correction by the data-assimilation system, compared to the signal that have been observed and corrected for some time already. Note especially that the standard deviation and the error in hv is higher at the jets' entry points to the observed area, compared to the rest of the observed area. Finally, we point out that the ensemble means for both hu and hv capture the main features of the truth in the eastern part of the domain as well, with a much lower error than in the first three experiments, even though this area is never observed. This is due to the transport of information that is assimilated into the system, along with the currents.

The drift trajectory forecast in Figure 14 is another indication of how well features are kept in the system even after the assimilation is ended. Drifter 24 starts close to the outflow of the southernmost jet in the western half of the domain, meaning that its underlying current has been influenced by the assimilation system for some time before the start of the forecast. This is reflected in the one-day forecast, which is almost as good as the forecast using all moorings, but the spread in the ensemble increases as we reach the two- and three-day forecasts. Most of the ensemble members still show the correct characteristics and therefore maintain the flow characteristics even without using further observations. The forecast also opens up for the possibility that the true drifter can turn north instead of south, but only with a very small probability, and we see that the ensemble mean trajectory is slightly south of the truth. For drifter number 2, however, row five of Figure 15 shows that the ensemble is quite confident that the drifter will move southwards. This drifter starts just on the outside of the observed area, and between the two dominating jets. Still, the forecast has a significantly lower spread than the experiments using all drifter observations. The forecast for this drifter turns out to be wrong, however, as a large majority of the ensemble trajectories are towards the southeast. The forecast does still leave a small probability for northwards drift, as we know to be the true trajectory.

5.2.6 Experiment F: Assimilating data from moorings in only the southern half of the domain

This time we use observations from the southern half of the domain, capturing only one of the initial jets. These observations are not sufficient to capture the true state, as can be seen from the obtained ensemble mean in the lower row of Figure 10. In the observed area, the mean is dominated by small-scale eddies that are not found in the truth, whereas the unobserved part of the domain hardly have any features at all. From the standard deviation and error plots in Figures 11 and 12, respectively, we see larger values than in any of the other experiments along the boundary of the observed area. This experiment illustrates better than the previous one how the ensemble needs to make larger adjustment on the ensemble members in the outskirts of the observed parts of the domain. Since there is limited information transport between the observed and unobserved areas, this experiment leads to larger errors than in Experiment E.

At the start of the drift trajectory forecast, drifter number 24 is just within the observed area, whereas drifter number 2 is just outside. Our choice of drifters should therefore not favour one of the half-domain experiments more than the other. Still, we see from Figures 14 and 15 that the forecasts produced by observations in the southern half of the domain have a much larger spread than the forecasts made after using observations in the western half. Particularly, the long-term trajectory characteristics of drifter 2 are very different from all previous experiments.

5.2.7 Comparison of forecast errors

To show that the forecast results for drifter 24 just discussed are reasonably representative for the majority of drifters, we investigate general forecasting statistics by defining a forecast error norm. First, let the error in the ensemble forecast for drifter d at time t^n be defined as

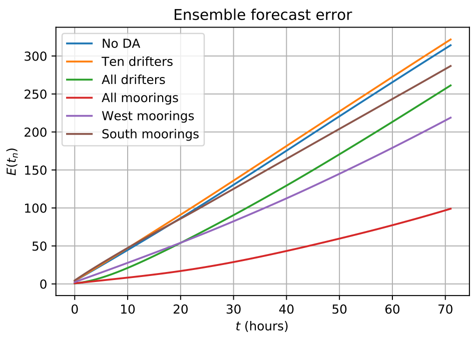
$$E_d(t^n) = \frac{1}{N_e} \sum_{i=1}^{N_e} \left[\left(x^n_{i,d} - x^n_{true,d} \right)^2 + \left(y^n_{i,d} - y^n_{true,d} \right)^2 \right]. \tag{48}$$

Furthermore, let the forecast error be the square root of the E^n_d mean over all drifters,

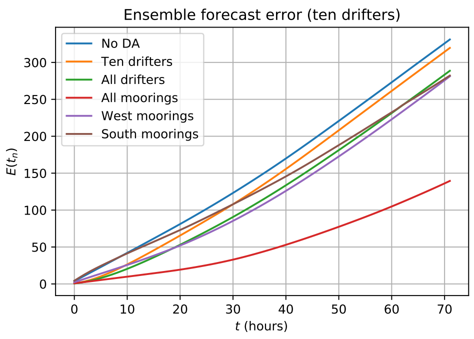
$$E(t^n) = \sqrt{\frac{1}{N_D} \sum_{d=1}^{N_D} E_d(t^n)}. \tag{49}$$

Similarly, we define the root-mean-square error $RMSE(t^n)$ in the same fashion as (48) and (49), but use the ensemble mean instead of the true drifter position. A low RMSE indicates low spread in the ensemble forecast, whereas a low error confirms that the true drifter location is within the low-spread forecast. On the contrary, if the RMSE is low, but the error is large, the ensemble gives a confident but wrong forecast.

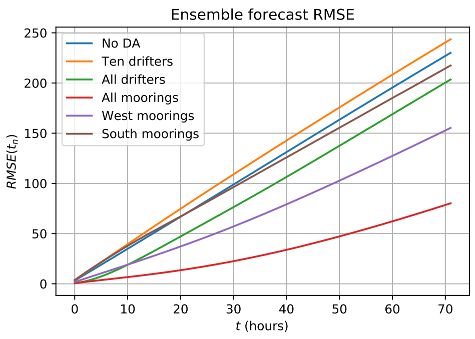
Figure 16 shows how the forecast error $E(t^n)$ and $RMSE(t^n)$ develop over time for the six different forecast experiments. The figures to the left (16a and 16c) shows the error when we consider all drifters, and the figures to the right (16b and 16d) consider only the ten drifters that are used in experiment B. The figures show that good results are consistently obtained by using observations from all the moorings, as the forecast error for this experiment is significantly lower than for the others. Also, we see that the error and the RMSE are consistent for all experiments, which means that the ensemble mean often is a good representation of the true drift trajectory. The exception is the west moorings experiment, which has a higher error than RMSE relative to the other experiments when considering the forecast for ten drifters only. The reason for this behavior is that the true trajectory often is forecasted as an ensemble outlier, such as seen for the west moorings experiment for drifter 2 in Figure 15. When comparing Figures 16c and 16d, we see that this behaviour is slightly over-represented among the ten selected drifters.



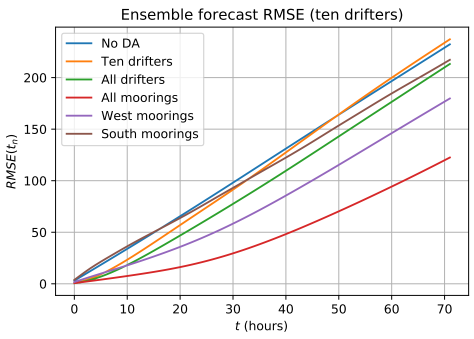
(a)



(b)



(c)



(d)

Figure 16: Mean forecast error for all six forecast experiments, considering the forecast for (a) all 64 drifters and (b) only the ten handpicked drifters used in experiment B. The second row shows RMSE as the comparable measure in terms of the forecast mean instead of the true drifter trajectory, again for (c) all 64 drifters and (d) the ten handpicked drifters only. The forecast errors are lowest for the experiment using observations from all moorings, whereas observations of some drifters do not improve the forecast much compared to the forecast without data assimilation.

In general, the forecasts are best when all moorings are observed, followed by observation of moorings in the west of the domain (runner-up for long-term forecast), and observations of all drifters (runner-up for short-term forecast). It is worth noting the differences between the two experiments that use observations from moorings in only half of the domain. Even though both experiments use the same kind and same number of observations, the west moorings enable a much better forecast as they observe a larger portion of the information flow over time.

Perhaps more interesting is the relationship between the forecast errors from using observations from ten drifters, and the forecast errors when using no observations. In Figure 16b, we see that using the drifters give a significantly better short-term forecast, whereas there is only a slight improvement on the long-term forecast. When the forecast for all 64 drifters are taken into account, however, we see that the long-term forecast becomes slightly worse by using these ten observations, compared to using no data assimilation at all. This could be due to our choice of local covariance structures, which is enforced on the ensemble through the IEWPF method. An observation is assimilated in the ensemble members through adding a dipole that gives the correct current at the drifter position. A side effect may be that the dipole induces a wrong current a small distance away from the drifter, causing the forecast for unobserved drifters at that location to be worse than if no data assimilation had been performed.

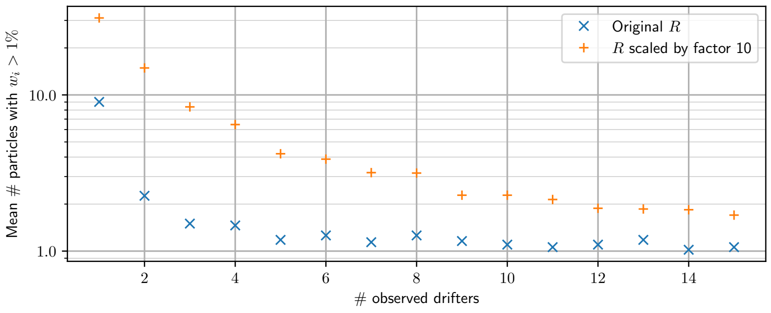


Figure 17: The number of particles guaranteed to be resampled when using a 100-member standard particle filter with residual resampling, for observations from different numbers of drifters. Blue crosses use the original weights based on the same uncertainty as the forecast experiments, and the weight is distributed on very few particles even for very low-dimensional observations. The yellow plus signs show the weight distributions assuming a tenfold increase in the observation covariance matrix R, but even with less reliable observations, the ensemble collapses for any more than six observed drifters.

5.3 Collapse of the standard particle filter

Collapse of the standard particle filter for high-dimensional observations is well known in the literature (see [3, 4, 5]). To illustrate its inefficiency, we look at the weight distribution using observations from a varying number of drifters.

Normalized weights are calculated using (6), which depends only on the size of the innovation $\boldsymbol{d}_i^n$ and the observation covariance matrix R, and not the size of the model error covariance matrix Q. Since our experiments start after a three day spin-up period, the ensemble has the highest variance during the initial data-assimilation cycles. In the IEWPF, this corresponds to a low target weight during the first iterations, while the ensemble is gradually adjusted according to the observations. With the standard particle filter, however, the large spread in the spin-up ensemble makes it very prone to collapse already in the first assimilation cycle. This is indeed what we observe, even with observations from only a single drifter.

To give the standard particle filter a fair chance, we run an experiment for three simulation days using the IEWPF method on observations from all 64 drifters, and thus obtain a well-distributed ensemble with a low spread and good representations of the underlying ocean current at the drifter locations. Under the restriction of fitting on a single commodity-level GPU, the ensemble size is kept as $N_e = 100$. The ensemble then runs to the next observation time, and we calculate the innovation vector using all drifters. We use subsets of the innovation vector to calculate normalized weights for different numbers of observed drifters. For each observation size, 50 drifter subsets are chosen at random, and for each subset we find the number of particles that have a normalized weight larger than $1/N_e$ (in this case, 1%), which guarantees that the given particle is kept in the ensemble when using residual sampling [26]. Figure 17 shows the mean number of particles which are guaranteed to be resampled for different observation sizes. The figure shows that if we observe one drifter only, we can expect about nine drifters to obtain a weight larger than $1/N_e$, but already when observing two drifters we see that this weight level reached by two particles only, which results in an ensemble collapse. Since the weight distribution depends largely on the size of the observation error, we make the same weight calculations assuming that the uncertainty in the observations are ten times as larger. The result is that the weight is distributed on more particles for all the observation sizes, but when observing six or more drifters, most of the weight is still on only three particles. This experiment clearly confirms how the standard particle filter cannot be used for the application at hand.

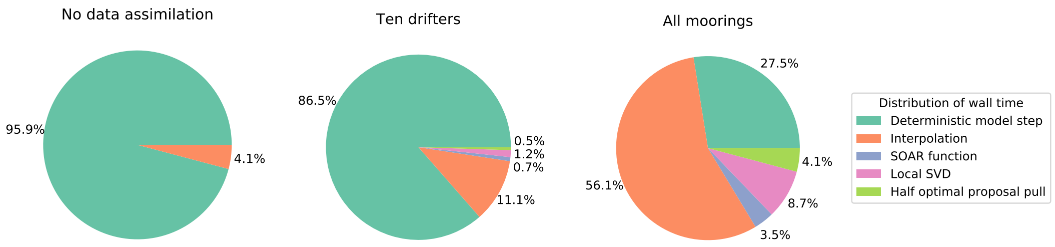


Figure 18: The pie charts show the distribution of total GPU compute time spent in the different CUDA kernels during the data-assimilation part of three chosen experiments. To the left, we see that only a small part of the compute time is spent on generating the model error, and that the time spent in the SOAR function is negligible. The center chart shows that the overhead from data assimilation on a small drifter set triples the amount of time spent in kernels that do not contribute to solving the deterministic model. With a large number of moorings, however, the majority of the time is spent in the interpolation kernel, and other kernels related to the data assimilation also play a significant part of the compute time, as seen to the right.

5.4 Computational performance

The baseline for evaluating the computational performance of the data-assimilation system is the efficiency of running just the model, consisting of the numerical scheme for solving the shallow-water equations. The implementation is based on the same approach as a GPU implementation of a very similar scheme [36], and has been profiled and optimized to maximize its performance and the occupancy of the GPU. The scheme has also been tuned to use the optimal block-size configuration applicable to the specific GPU used in this work, an Nvidia GeForce GTX 780. The efficiency of all other kernels will be evaluated through a comparison to the deterministic model step, to search for limitations and bottlenecks for relevant applications, such as the experiments in Section 5.2.

We start by evaluating the computational performance of the stochastic model errors by comparing the run-time required for generating $\boldsymbol{\beta}$ and deterministically evolving the model one time step. We analyse a benchmark application using 500×300 grid cells, with model errors added every Δt_{scheme} and a coarsening factor similar to the above experiments, $c_\Omega = 5$. Profiling reveals that 74% of the GPU compute time is spent evaluating the numerical scheme, 22% is spent on interpolation, 1.4% on the SOAR function, and 1.4% on generating random numbers. This indicates that the implementation of the model error is sufficiently efficient compared to the model step and does not represent a major performance bottleneck. It should also be noted that whereas the relationship between the deterministic model step and the interpolation does not change when the number of grid cells is increased, the relative amount of compute time spent in the other two kernels becomes negligible.

Figure 18 shows the distribution of GPU compute time during some data-assimilation cycles for three of the experiments from Section 5.2, restricted to $N_e = 10$ to make it feasible to run short experiments through the profiler. During these experiments, the model error is added every model time step, which typically consists of eight steps of the numerical scheme. In the experiment with no data assimilation, the model error amounts to only 4.1% of the GPU compute time. With assimilation of ten drifters, shown in the center pie chart, the fraction of time spent on interpolation increases to 11.1%, whereas an additional 2.4% is spent on other assimilation-related kernels. The majority of the time is nevertheless still spent on the deterministic model step, meaning that there is limited value in optimizing the particle filter kernels for this problem size. When assimilating all 240 moorings, we reach a situation in which the interpolation represents 56.1% of the GPU compute time, and a further effort in optimizing the IEWPF implementation should be considered. Some ideas for this are discussed at the end of this section.

Figure 19 shows the wall clock time for each of the six experiments with $N_e = 100$, normalized with respect to the experiment without data assimilation. Note that the additional time spent on the data assimilation per drifter observed is constant for the mooring experiments. For the drifters, there is a small overhead with 64 drifters, but for ten drifters the data assimilation takes almost twice as long per drifter as for the mooring experiments. These observations are well in accordance with the algorithmic complexity outlined in Figure 3.

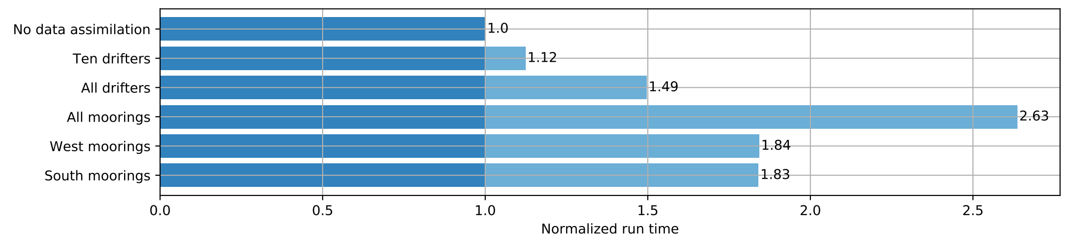


Figure 19: Wall clock run-time measured for the data-assimilation part for each of the six forecast experiments, normalized with respect to the experiment without data assimilation. The lighter color indicates time used on the data assimilation. The assimilation of observations from ten drifters gives a 12% overhead, whereas using all 240 moorings adds 160% to the total wall clock time.

The wall clock run-time for simulating one hour of data assimilation, consisting of twelve data-assimilation cycles for 100 particles, is 41 seconds on the Nvidia GeForce GTX 780. This GPU represents a commodity-level graphics card, which has been used for five years at the time of writing, and thereby represents a class of GPUs that is widely available. By upgrading to a modern high-end GPU, such as the Nvidia Tesla P100, we have observed a 3 times speed up without adjusting any implementation configurations.

The profiler shows that the occupancy (the concurrent utilization of the available resources on the GPU) is 97.3% for sufficiently large domains, without exposing any clear strategy for further optimization. At this point, the kernel has already been tuned by balancing occupancy and register spilling to achieve optimal performance. To increase the performance for the experiments with a large number of observations, the main focus should therefore be on optimizing the use of the bicubic interpolation kernel. A high-level performance optimization would be to introduce parallel processing of drifters during the optimal proposal pull, as the interpolation is currently done once per drifter during this step. This would require that drifters with the same offset configuration are identified, and that those drifters are color coded according to their location within the domain to avoid overlapping memory access. For this strategy to be fruitful, the number of drifters must be sufficiently large compared to the number of possible offset configurations, c_Ω^2, so that the extra computational work required to color code the drifters is compensated by the expected amount of increased parallelization. This trade-off is less of an issue with mooring observations, as the constant location of the moorings would mean that the color coding can be pre-computed, rather than updated for every observation time step.

6 Summary and conclusions

We have presented a GPU implementation of the state-of-the-art implicit equal-weights particle filter applied to an ensemble of simplified ocean models and used it to forecast drift trajectories. The observations are obtained from the positions of passive drifters and direct ocean current measurements from moored buoys in a synthetic true state. Forecasts of drift trajectories have been generated for a near-realistic unstable jet experiment, for which the instabilities develop chaotically due to random model error realizations. All parts of the data-assimilation system (model, model errors, and particle filter) have been designed to take advantage of fine-grained data parallelism, and we have shown that the most computationally expensive components of the system are able to efficiently utilize the resources on a GPU.

We have shown how the forecast quality is improved as more drifter and mooring observations are assimilated through the forecast experiments. The best results are achieved when information is assimilated from all 240 available moorings equally distributed throughout the domain. Even though the observations cover only approximately 0.1% of the state space, the ensemble mean at the start of the forecast is a very good representation of the true state. Since the ensemble contains a very accurate description of the true ocean currents, the forecast is shown to be both accurate and confident, even in the long-term up to three days. Two of the experiments assimilated mooring observations from only the southern half or only the western half of the domain, respectively. As the dominating currents are in the east-west direction, these experiments illustrate the importance of considering information transport in the system. The ensemble mean after the data-assimilation period and the general quality of the drift trajectory forecasts are significantly better when both the jets were partially observed (west moorings) compared to observations of one full jet (south moorings) only.

With fewer drifter observations, we have seen that the ensemble is not able to capture the model state with the same accuracy compared to using lots of moorings. However, the short-term forecasts are significantly improved for the first 12 hours, which is an important time scale for search and rescue operations. The drifter experiments are also more realistic in terms of equipment than the mooring experiments. In an operational setting, drifters could be released in the area of interest by, e.g., a search and rescue vessel, to sample relevant observations. With our approach consisting of an efficient data-assimilation system applied to simplified models, these observations can be used to perform in-situ drift trajectory forecasts, using the most recent traditional ocean forecasts as starting points.

Although the results from the particle filter are good, some issues remain. Since we assimilate single-point mass transport, the update takes a dipolar structure in sea-surface height around the observation location. The size of these dipoles is limited to the length scales in the model error covariances and can be smaller than the length scale of actual eddies, potentially leading to unrealistic updates some distance away from the observation localtions. This is indeed what we see when only 10 drifters are present. Different structures for the model errors should improve this issue.

All experiments are conducted with a barotropic ocean model. The resulting currents would not be representative of realistic situations with strong bottom topography, and hence a reduced gravity set up would be more appropriate. This is not conceptually different from our current approach and, since this also would allow us to use a larger time step, it could further contribute to accelerate the model forecasts. A more extensive alternative to a reduced gravity model would be to extend our method to multilayered systems. But again, no major obstacles are expected for such an extension. In fact, it might result in a better balance between data assimilation effort and forecast effort.

Acknowledgments

HHH and MLS thanks the Research Council of Norway for funding the GPU Ocean project, with grant number 250935. PJvL thanks the European Research Council for funding the CUNDA grant 694509 under the European Union's Horizon 2020 research and innovation programme. Some of the computations were performed on resources provided by UNINETT Sigma2 – the National Infrastructure for High Performance Computing and Data Storage in Norway under project number nn9550k. Furthermore, the authors would like to thank André Rigland Brodtkorb for valuable discussions, and Kai Håkon Christensen and Knut-Andreas Lie for feedback on the manuscript.

Supplementary material

The source code for the methods and experiments described in this paper is available under an GNU open source license under the DOI 10.5281/zenodo.3458291. The complete datasets representing the ensemble results presented in this paper are available under a GNU free and open source license under the DOI 10.5281/zenodo.3457538.

A A modified implicit equal-weights particle filter

As mentioned in the main text, the update equation for each particle $\boldsymbol{\psi}_i$ in the original implicit equal-weights particle filter (IEWPF) [29] is

$$\boldsymbol{\psi}_i^n = \boldsymbol{\psi}_i^{n,a} + \alpha_i^{1/2} P^{1/2} \xi_i. \tag{A.1}$$

Because $\boldsymbol{\psi}_i^{n,a}$ is a deterministic move of the particles according to (11), this is a transformation of coordinates from $\boldsymbol{\psi}$ to ξ, so we can write

$$q(\boldsymbol{\psi}^n | \boldsymbol{\psi}_{1:N_e}^{n-1}, \boldsymbol{y}^n) = \frac{q(\xi)}{\left\| \frac{\mathrm{d}\boldsymbol{\psi}}{\mathrm{d}\xi} \right\|}. \tag{A.2}$$

The denominator represents the absolute value of the determinant of the Jacobian, and can be found through the mapping between ξ_i and $\boldsymbol{\psi}_i^n$. This mapping is complicated because α_i also depends on ξ_i, but in an up-to-now unknown way. Using (A.2), the expression for the weights from (10) becomes

$$w_i^n = \frac{p(\boldsymbol{y}^n|\boldsymbol{\psi}_i^n)p(\boldsymbol{\psi}_i^n|\boldsymbol{\psi}_i^{n-1})}{N_e p(\boldsymbol{y}^n)q(\xi)} \left\| \frac{\mathrm{d}\boldsymbol{\psi}_i^n}{\mathrm{d}\xi_i} \right\|. \tag{A.3}$$

By assuming that α_i only depends on ξ_i through its magnitude $\xi_i^T\xi_i = \gamma_i$, (A.3) can be written as the scalar implicit equation

$$-\log\left(w_i^n\right) = (\alpha_i - 1)\gamma_i - 2\log\left[\alpha_i^{N_\psi/2}\left|1 + \frac{\gamma_i}{\alpha_i^{1/2}}\frac{\partial\alpha_i^{1/2}}{\partial\gamma_i}\right|\right] + c_i, \tag{A.4}$$

in which

$$c_i = \phi_i - \log\left(w_i^{n-1}\right) \tag{A.5}$$

and

$$\phi_i = (\boldsymbol{d}_i^n)^T\left(HQH^T + R\right)^{-1}\boldsymbol{d}_i^n. \tag{A.6}$$

The essence of the IEWPF is that in order to ensure a significant weight for all particles, α_i is chosen so that all weights become equal to a target weight, $w_i^n = w_{target}$ for $i = 1, ..., N_e$, leading to a nonlinear equation for each α_i. See [29] or the appendix of Skauvold et al. [31] for further details.

It is important to set the target weight such that all particles can reach it. Since a smaller c_i leads to a larger weight, and since c_i denotes the best value for the weight that particle i can attain, the target weight has to be related to the maximum of the c_i, and it is chosen as

$$w_{target} = \max_{i=1,\ldots,N_e}\{c_i\}. \tag{A.7}$$

By setting $-\log(w_i) = w_{target}$ in (A.4), the expression for α_i becomes

$$(\alpha_i - 1)\gamma_i - 2\log\left[\alpha^{N_\psi/2}\left|1 + \frac{\gamma_i}{\alpha_i^{1/2}}\frac{\partial\alpha_i^{1/2}}{\partial\gamma_i}\right|\right] = w_{target} - c_i. \tag{A.8}$$

This equation is equivalent to

$$\Gamma\left(\frac{N_x}{2}, \frac{\alpha_i\gamma_i}{2}\right) = e^{-c_i^\star/2}\Gamma\left(\frac{N_x}{2}, \frac{\gamma_i}{2}\right), \tag{A.9}$$

which can be solved numerically for α_i by, e.g., the Newton method, as illustrated by Skauvold et al. [31]. Here, we use that

$$c_i^\star = w_{target} - c_i = \max_{j=1,\ldots,N_e}\{c_j\} - c_i, \tag{A.10}$$

and $\Gamma(s, x) = \int_0^x t^{s-1}e^{-t}\mathrm{d}t$ is the incomplete lower gamma function. Whenever the state space N_ψ is large, however, (A.9) becomes harder to solve as the gamma functions become prone to overflow. In this high-dimensional limit, it is possible to solve (A.8) analytically in terms of the Lambert W function, as showed by Zhu et al. [29], as

$$\alpha_i = -\frac{N_\psi}{\gamma_i}W_0\left[-\frac{\gamma_i}{N_\psi}e^{-\gamma_i/N_\psi}e^{-c_i^\star/N_\psi}\right]. \tag{A.11}$$

As pointed out by Skauvold et al [31], only solutions $\alpha_i < 1$ should be accepted, meaning that only the zero branch for the Lambert W function is considered.

Two weaknesses of the scheme above can be identified. Firstly, for low-dimensional systems it can be shown that the posterior variance is always underestimated. The other weakness occurs for high-dimensional systems. Since all particles have to reach the same target weight, and that target weight has to be chosen as the weight of the weakest particle, the more particles we use the worse the weakest particle will be, so the further away all particles are pushed from the high likelihood values. So, in high dimensions, although the scheme is useful for small ensemble sizes, it degenerates at larger ensemble sizes.

To overcome the above-mentioned challenges, a revised two-stage IEWPF scheme has been proposed [31], which explores the complete proposal density and does not underestimate the posterior variance. The new update equation is

$$\psi_i^n = \psi_i^{n,a} + \beta^{1/2} P^{1/2} \nu_i + \alpha_i^{1/2} P^{1/2} \xi_i, \tag{A.12}$$

in which ν_i is a second random vector $\nu_i \sim N(0, I)$, and β is a covariance scaling parameter common to all particles. Using the same assumptions as for the one-stage method, (A.3) can now be written as

$$\log\left(w_i^n\right) = (\alpha_i - 1)\gamma_i + 2\beta^{1/2}\alpha_i^{1/2}\xi_i^T \nu_i + (\beta - 1)\zeta_i - 2\log\left[\alpha_i^{N_\psi/2}\left|1 + \frac{\gamma_i}{\alpha_i^{1/2}}\frac{\partial\alpha_i^{1/2}}{\partial\gamma_i}\right|\right] + c_i, \tag{A.13}$$

in which c_i is according to (A.5), and $\zeta_i = \nu_i^T \nu_i$. To solve (A.13), ν_i is constructed to be perpendicular to ξ_i, making the cross term between the two random vectors disappear. In the case of large N_ψ, and by defining

$$c_i^\star = w_{target} - c_i - (\beta - 1)\zeta_i, \tag{A.14}$$

(A.13) becomes similar to (A.8), with solution according to (A.11). We require that $c_i^\star \geq 0$, which is equivalent to

$$\beta \leq \frac{w_{target} - c_i}{\zeta_i} + 1. \tag{A.15}$$

This equation shows that the introduction of β allows us to choose a different target weight. By choosing the target weight to be $w_{target} = \overline{c_i}$, the mean of c_i across the ensemble, β can be set to the minimum value of the right-hand-side of (A.15),

$$\beta = \min_{i=1,\ldots,N_e}\left\{\frac{\overline{c_i} - c_i}{\zeta_i} + 1\right\}. \tag{A.16}$$

Since $\overline{c} - c_i \approx N_y \pm \sqrt{2N_y}$ and $\zeta_i \approx N_\psi \pm \sqrt{2N_\psi}$, the parameter $\beta^{1/2}$ should remain real as long as $N_\psi >> N_y$, which holds for our high-dimensional application.

Choosing the target weight equal to the mean of c_i, is equivalent to choosing it equal to the mean of the optimal proposal weights. An advantage with this choice is that the target weight will not vary much when N_e increases. This is contrary to the one-stage scheme, in which the target weight is equal to $\max_{i=1,..,N_e}\{c_i\}$, which becomes larger if N_e increases. In other words, the one-stage scheme pushes the particles further and further away from the high-probability regions of the posterior. Because of its choice of w_{target}, the two-stage scheme does not have this problem, and is the method of choice in this paper.

References

[1] K.-F. Dagestad, J. Röhrs, Ø. Breivik, and B. Ådlandsvik, "OpenDrift v1.0: a generic framework for trajectory modelling," *Geoscientific Model Development*, vol. 11, no. 4, pp. 1405–1420, 2018.

[2] 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.

[3] C. Snyder, T. Bengtsson, P. Bickel, and J. Anderson, "Obstacles to high-dimensional particle filtering," *Monthly Weather Review*, vol. 136, no. 12, pp. 4629–4640, 2008.

[4] P. J. van Leeuwen, "Particle filtering in geophysical systems," *Monthly Weather Review*, vol. 137, no. 12, pp. 4089–4114, 2009.

[5] C. Snyder, T. Bengtsson, and M. Morzfeld, "Performance bounds for particle filters using the optimal proposal," *Monthly Weather Review*, vol. 143, no. 11, pp. 4750–4761, 2015.

[6] P. J. van Leeuwen, "Nonlinear data assimilation in geosciences: an extremely efficient particle filter," *Quarterly Journal of the Royal Meteorological Society*, vol. 136, no. 653, pp. 1991–1999, 2010.

[7] J. Poterjoy, R. A. Sobash, and J. L. Anderson, "Convective-scale data assimilation for the weather research and forecasting model using the local particle filter," *Monthly Weather Review*, vol. 145, no. 5, pp. 1897–1918, 2017.

[8] P. Van Leeuwen, L. Nerger, R. Potthast, S. Reich, , and H. Kunsch, "A review of particle filters for geoscience applications," *Quarterly Journal of the Royal Meteorological Society*, 2019.

[9] F. Lopez, L. Zhang, A. Mok, and J. Beaman, "Particle filtering on GPU architectures for manufacturing applications," *Computers in Industry*, vol. 71, pp. 116–127, 2015.

[10] A. Gelencsér-Horváth, G. J. Tornai, A. Horváth, and G. Cserey, "Fast, parallel implementation of particle filtering on the GPU architecture," *EURASIP Journal on Advances in Signal Processing*, vol. 2013, p. 148, Sep 2013.

[11] L. M. Murray, "Bayesian state-space modelling on high-performance hardware using LibBi," *arXiv e-prints*, Jun 2013.

[12] F. Bai and X. Hu, "Cloud MapReduce for particle filter-based data assimilation for wildfire spread simulation," in *Proceedings of the High Performance Computing Symposium*, HPC '13, (San Diego, CA, USA), pp. 11:1–11:6, Society for Computer Simulation International, 2013.

[13] F. Bai, F. Gu, X. Hu, and S. Guo, "Particle routing in distributed particle filters for large-scale spatial temporal systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, pp. 481–493, Feb 2016.

[14] T. Blattner and S. Yang, "Performance study on CUDA GPUs for parallelizing the local ensemble transformed Kalman filter algorithm," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 2, pp. 167–177, 2012.

[15] S.-C. Wei and B. Huang, "A GPU-accelerated extended Kalman filter," in *High-Performance Computing in Remote Sensing* (B. Huang and A. J. Plaza, eds.), vol. 8183, pp. 35 – 42, International Society for Optics and Photonics, SPIE, 2011.

[16] J. C. Quinn and H. D. Abarbanel, "Data assimilation using a GPU accelerated path integral Monte Carlo approach," *Journal of Computational Physics*, vol. 230, no. 22, pp. 8168–8178, 2011.

[17] V. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey, "Debunking the 100x GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU," in *ISCA '10: Proceedings of the 37th annual international symposium on Computer architecture*, (New York, NY, USA), pp. 451–460, ACM, 2010.

[18] A. Apte, C. K. R. T. Jones, and A. M. Stuart, "A Bayesian approach to Lagrangian data assimilation," *Tellus A: Dynamic Meteorology and Oceanography*, vol. 60, no. 2, pp. 336–347, 2008.

[19] E. T. Spiller, A. Apte, and C. K. R. T. Jones, "Assimilating en-route Lagrangian observations," *Tellus A: Dynamic Meteorology and Oceanography*, vol. 65, no. 1, pp. 1–14, 2013.

[20] E. T. Spiller, A. Budhiraja, K. Ide, and C. K. Jones, "Modified particle filter methods for assimilating Lagrangian data into a point-vortex model," *Physica D: Nonlinear Phenomena*, vol. 237, no. 10, pp. 1498–1506, 2008. Perspectives in Fluid Dynamics.

[21] L. Kuznetsov, K. Ide, and C. K. R. T. Jones, "A method for assimilation of Lagrangian data," *Monthly Weather Review*, vol. 131, no. 10, pp. 2247–2260, 2003.

[22] A. Apte and C. K. R. T. Jones, "The impact of nonlinearity in Lagrangian data assimilation," *Nonlinear Processes in Geophysics*, vol. 20, no. 3, pp. 329–341, 2013.

[23] M. J. Carrier, H. Ngodock, S. Smith, G. Jacobs, P. Muscarella, T. Ozgokmen, B. Haus, and B. Lipphardt, “Impact of assimilating ocean velocity observations inferred from Lagrangian drifter data using the NCOM-4DVAR,” *Monthly Weather Review*, vol. 142, no. 4, pp. 1509–1524, 2014.

[24] L. Slivinski, E. Spiller, A. Apte, and B. Sandstede, “A hybrid particle-–ensemble Kalman filter for Lagrangian data assimilation,” *Monthly Weather Review*, vol. 143, no. 1, pp. 195–211, 2015.

[25] L. Slivinski, L. Pratt, I. Rypina, M. Orescanin, B. Raubenheimer, J. MacMahan, and S. Elgar, “Assimilating Lagrangian data for parameter estimation in a multiple-inlet system,” *Ocean Modelling*, vol. 113, pp. 131–144, 2017.

[26] J. S. Liu and R. Chen, “Sequential Monte Carlo methods for dynamic systems,” *Journal of the American Statistical Association*, vol. 93, pp. 1032–1044, 1998.

[27] A. Doucet, S. Godsill, and C. Andrieu, “On sequential Monte Carlo sampling methods for Bayesian filtering,” *Statistics and Computing*, vol. 10, pp. 197–208, Jul 2000.

[28] M. Ades and P. J. van Leeuwen, “An exploration of the equivalent weights particle filter,” *Quarterly Journal of the Royal Meteorological Society*, vol. 139, no. 672, pp. 820–840, 2013.

[29] M. Zhu, P. J. van Leeuwen, and J. Amezcua, “Implicit equal-weights particle filter,” *Quarterly Journal of the Royal Meteorological Society*, vol. 142, no. 698, pp. 1904–1919, 2016.

[30] A. J. Chorin, M. Morzfeld, and X. Tu, *A survey of implicit particle filters for data assimilation*, pp. 63–88. New York, NY: Springer New York, 2013.

[31] J. Skauvold, J. Eidsvik, P. J. van Leeuwen, and J. Amezcua, “A revised implicit equal-weights particle filter,” *Quarterly Journal of the Royal Meteorological Society*, vol. 145, no. 721, pp. 1490–1502, 2019.

[32] G. Madec, R. Bourdallé-Badie, P.-A. Bouttier, C. Bricaud, D. Bruciaferri, D. Calvert, J. Chanut, E. Clementi, A. Coward, D. Delrosso, C. Ethé, S. Flavoni, T. Graham, J. Harle, D. Iovino, D. Lea, C. Lévy, T. Lovato, N. Martin, S. Masson, S. Mocavero, J. Paul, C. Rousset, D. Storkey, A. Storto, and M. Vancoppenolle, “NEMO ocean engine,” Oct 2017.

[33] K. H. Christensen, Ø. Breivik, K.-F. Dagestad, J. Röhrs, and B. Ward, “Short-term predictions of oceanic drift,” *Oceanography*, vol. 31, pp. 59–67, September 2018.

[34] R. J. LeVeque, *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2004.

[35] T. R. Hagen, M. O. Henriksen, J. M. Hjelmervik, and K.-A. Lie, *How to solve systems of conservation laws numerically using the graphics processor as a high-performance computational engine*, pp. 211–264. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[36] A. R. Brodtkorb, M. L. Sætra, and M. Altinakar, “Efficient shallow water simulations on GPUs: Implementation, visualization, verification, and validation,” *Computers & Fluids*, vol. 55, no. 0, pp. 1–12, 2012.

[37] M. de la Asunción, J. Mantas, and M. Castro, “Simulation of one-layer shallow water systems on multicore and CUDA architectures,” *The Journal of Supercomputing*, vol. 58, pp. 206–214, Nov 2011.

[38] H. Meuer, E. Strohmaier, J. Dongarra, and H. Simon, “Top 500 supercomputer sites.” `http://www.top500.org/`, November 2018.

[39] H. Sutter, “The free lunch is over: A fundamental turn toward concurrency in software,” *Dr. Dobb’s Journal*, vol. 30, no. 3, pp. 202–210, 2005.

[40] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 2010.

[41] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih, “PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation,” *Parallel Computing*, vol. 38, no. 3, pp. 157–174, 2012.

[42] T. Oliphant, *Guide to NumPy*. Trelgol Publishing, 2006.

[43] J. Hunter, “Matplotlib: A 2d graphics environment,” *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[44] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, and Jupyter development team, “Jupyter notebooks – a publishing format for reproducible computational workflows,” in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pp. 87–90, 2016.

[45] A. Chertock, M. Dudzinski, A. Kurganov, and M. Lukácová-Medvidová, “Well-balanced schemes for the shallow water equations with Coriolis forces,” *Numerische Mathematik*, Dec 2017.

[46] H. Holm and A. Brodtkorb, “Adapting a two-dimensional finite volume scheme for real-world oceanographic simulations,” 2019. [preprint].

[47] S. Hatfield, A. Subramanian, T. Palmer, and P. Düben, “Improving weather forecast skill through reduced-precision data assimilation,” *Monthly Weather Review*, vol. 146, no. 1, pp. 49–62, 2018.

[48] M. Harris, “Optimizing parallel reduction in CUDA,” tech. rep., Nvidia Developer Technology, 2007.

[49] J. Galewsky, R. K. Scott, and L. M. Polvani, “An initial-value problem for testing numerical models of the global shallow-water equations,” *Tellus A: Dynamic Meteorology and Oceanography*, vol. 56, no. 5, pp. 429–440, 2004.

[50] T. M. Hamill, “Interpretation of rank histograms for verifying ensemble forecasts,” *Monthly Weather Review*, vol. 129, no. 3, pp. 550–560, 2001.

Paper IV

Real-World Oceanographic Simulations on the GPU using a Two-Dimensional Finite Volume Scheme

André Rigland Brodtkorb, Håvard Heitlo Holm

Preprint: arXiv:1912.02457, 2019
In review

Real-World Oceanographic Simulations on the GPU using a Two-Dimensional Finite-Volume Scheme

André R. Brodtkorb*[1,2] and Håvard Heitlo Holm[3,4]

[1] Norwegian Meteorological Institute, P.O. Box 43 Blindern, NO-0313 Oslo, Norway.
[2] Oslo Metropolitan University, Department of Computer Science, P.O. Box 4 St. Olavs plass, NO-0130 Oslo, Norway.
[3] SINTEF Digital, Mathematics and Cybernetics, P.O. Box 124 Blindern, NO-0314 Oslo, Norway.
[4] Norwegian University of Science and Technology, Department of Mathematical Sciences, NO-7491 Trondheim, Norway.

Abstract

In this work, we take a modern high-resolution finite-volume scheme for solving the rotational shallow-water equations and extend it with features required to run real-world ocean simulations. Our contributions include a spatially varying north vector and Coriolis term required for large scale domains, moving wet-dry fronts, a static land mask, bottom shear stress, wind forcing, boundary conditions for nesting in a global model, and an efficient model reformulation that makes it well-suited for massively parallel implementations. Our model order is verified using a grid convergence test, and we show numerical experiments using three different sections along the coast of Norway based on data originating from operational forecasts run at the Norwegian Meteorological Institute. Our simulation framework shows perfect weak scaling on a modern P100 GPU, and is capable of providing tidal wave forecasts that are very close to the operational model at a fraction of the cost. All source code and data used in this work are publicly available under open licenses.

1 Introduction

The aim of this paper is to simulate realistic oceanographic scenarios using a modern finite-volume scheme on GPUs. Modern operational ocean models, such as the Regional Ocean Modeling System (ROMS) [1] and the Nucleus for European Modelling of the Ocean (NEMO) [2], are based on solving the primitive equations that describe conservation of mass, momentum, temperature, and salinity. Even though these ocean models are heavily optimized, their complex physical and mathematical properties make them very computationally demanding. The NorKyst-800 model system [3], for example, is based on the ocean model ROMS and covers the Norwegian coast with 800 m horizontal resolution. It is run operationally by the Norwegian Meteorological Institute on a daily basis. The complete model consists of 42 vertical layers with 2600×900 grid cells each, and a 24 hour forecast takes 45 minutes when using 256 CPUs [4]. It is unfeasible to use such models for flexible on-demand simulations during search-and-rescue operations, oil spills, heavy lifting at sea, etc. In these cases, fast on-demand simulations based on the shallow-water equations can have a significant value for decision makers.

We simulate the shallow-water equations in a rotating frame of reference, targeting efficient simulation of ocean currents based on the operational NorKyst-800 data. Our model equations are essentially two-dimensional simplifications of the primitive equations and assume constant density and a vertically integrated momentum (see more detailed derivations in, e.g., Røed [5]). Our work starts with the recent high-resolution finite-volume scheme by Chertock et al. [6], and extends this scheme to support bottom shear stress, wind forcing, projection-free spatial variations in the north vector, spatially varying Coriolis parameter, land mask, and a moving wet-dry boundary. We present how to reconstruct a piecewise bilinear bathymetry from the piecewise constant bathymetry in the NorKyst-800 data, nesting

*Corresponding author: Andre.Brodtkorb@met.no of the model into the NorKyst-800 data using linearly interpolated boundary conditions, and generation of both higher and lower spatial resolution of the initial conditions.

We simulate increasingly challenging scenarios along the Norwegian coast, and compare with reference results from the operational model. As the shallow-water equations are particularly well suited for tidal and storm-surge predictions, we also compare our tidal-wave predictions at several locations with the reference dataset and observations. All datasets and source code used in this work are publicly available under open licenses.[1]

The rest of the paper is organized as follows: Section 2 gives a high-level description of the shallow-water equations and how they can be solved numerically. In Section 3, we present all the extensions that are required to tackle relevant oceanographic problems. The performance and numerical accuracy of our scheme are assessed in Section 4, where we show perfect weak scaling of the GPU implementation and run a grid convergence test with the new terms introduced in the scheme. We present operational-grade simulations in Section 5, and finally summarize the work in Section 6.

2 Mathematical formulation

We use the shallow-water equations with source terms to simulate ocean dynamics. Since the equations are purely barotropic, they are not expected to generate or preserve features such as mesoscale eddies in the long term. These features arise from baroclinic instabilities, typically caused by variations in temperature and salinity that are not included in the mathematical model, but the equations are nevertheless capable of capturing the most important physics required to simulate short-term ocean dynamics. We can write the equations with source terms as

$$\begin{aligned}\begin{bmatrix} h \\ hu \\ hv \end{bmatrix}_t + \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{bmatrix}_x + \begin{bmatrix} hv \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix}_y \\ = \begin{bmatrix} 0 \\ ghH_x \\ ghH_y \end{bmatrix} + \begin{bmatrix} 0 \\ -ru\sqrt{u^2+v^2}/h \\ -rv\sqrt{u^2+v^2}/h \end{bmatrix} + \begin{bmatrix} 0 \\ fhv \\ -fhu \end{bmatrix} + \begin{bmatrix} 0 \\ \tau_x \\ \tau_y \end{bmatrix}.\end{aligned} \tag{1}$$

Here, h is water depth, hu momentum along the x-axis, and hv momentum along the y-axis (see also Figure 1). Furthermore, g is the gravitational acceleration, H is the equilibrium ocean depth measured from a reference sea-surface level, r is the friction coefficient, f is the Coriolis parameter, and τ_x and τ_y are the wind stress along the x- and y-axis, respectively. In vector form, we can write the equations as

$$Q_t + F(Q)_x + G(Q)_y = B(Q) + S(Q) + C(Q) + W(Q). \tag{2}$$

Here, $Q = [h, hu, hv]^T$ is the vector of conserved variables and B, S, C, and W are source terms representing varying bathymetry, bed shear stress, Coriolis force, and wind drag, respectively.

By using a finite-volume discretization on a Cartesian grid to solve (2), we get the following semi-discrete formulation

$$\frac{\partial Q}{\partial t} = -\frac{1}{\Delta x}\left[F_{i+1/2,j} - F_{i-1/2,j}\right] - \frac{1}{\Delta y}\left[G_{i,j+1/2} - G_{i,j-1/2}\right] + B_{i,j} + S_{i,j} + C_{i,j} + W_{i,j}. \tag{3}$$

To evolve the solution in time, we use a second-order strong stability-preserving Runge-Kutta scheme [8],

$$\begin{aligned} Q^*_{i,j} &= Q^n_{i,j} + \Delta t M(Q^n)_{i,j} \\ Q^{n+1}_{i,j} &= \tfrac{1}{2}\Big(Q^n_{i,j} + \Big(Q^*_{i,j} + \Delta t M(Q^*)_{i,j}\Big)\Big), \end{aligned} \tag{4}$$

[1]The source code used to create plots and results in this article is available in revision 7281301a3a286351b6e857add12c2fbd0165b229 of the gpu-ocean repository, https://github.com/metno/gpu-ocean, released under GPL version 3. The plots are created using Jupyter Notebooks [7] in the folder demos/realisticSimulations, and the simulator code itself is under the directory SWESimulators as a Python module. We have used the NorKyst-800 forecast for 2019-07-16 from the Norwegian Meteorological Institute in this work, available on https://thredds.met.no/thredds/dodsC/fou-hi/norkyst800m-1h/NorKyst-800m_ZDEPTHS_his.an.2019071600.nc. The observed sea-surface levels are obtained from Se Havnivå, https://www.kartverket.no/sehavniva/, provided by the Norwegian Mapping Authority, Hydrographic Service. The data sets are available under the Creative Commons 4.0 BY International license as described on https://www.met.no/en/free-meteorological-data/Licensing-and-crediting and https://www.kartverket.no/en/data/Terms-of-use/.

in which M represents the right hand side of (3). The time step is restricted by the CFL condition given by

$$\Delta t \leq \frac{1}{4}\min\left\{\frac{\Delta x}{\max\left|u \pm \sqrt{gh}\right|}, \frac{\Delta y}{\max\left|v \pm \sqrt{gh}\right|}\right\}, \tag{5}$$

if we assume no Coriolis and wind drag. The dominant term for oceanographic applications is typically the speed of gravity waves $\sqrt{gh}$, and we use a conservative Courant number less than one in our simulations to account for Coriolis and wind drag.

2.1 Numerical scheme

The starting point for our simulator is the scheme proposed by Chertock et al. [6], from now on referred to as CDKLM. It is a modern central-upwind scheme that uses a reconstruction of the physical variables to be able to capture steady-state solutions important for physical oceanography, and includes source terms for varying bathymetry and Coriolis. The original paper includes several idealized test cases and examples demonstrating that the steady states in geostrophic balance are well captured. However, there are several shortcomings that need to be addressed before the scheme can be used for real-world simulations.

The CDKLM scheme is based on the same principles as the MUSCL scheme [9] to obtain a high-order, total variation diminishing discretization. For each cell in the domain, we first reconstruct a piecewise (linear) function for our physical variables (see Figure 1). At the interface between two cells, we evaluate the slope limited function from the right and left cell, and use these two values to compute the flux across the interface. The scheme uses the Riemann-solver-free central-upwind flux function,

$$F_{i+1/2} = \frac{a^+ F(Q^l_{i+1/2}) - a^- F(Q^r_{i+1/2})}{a^+ - a^-} + \frac{a^+ a^-}{a^+ - a^-}\left[Q^r_{i+1/2} - Q^l_{i+1/2}\right], \tag{6}$$

to evaluate the numerical flux [10]. Here, a^+ and a^- are the largest positive and negative wave speeds at the interface, respectively.

When adding source terms, it becomes challenging to preserve steady states such as “lake at rest”, i.e.,

$$\frac{\partial Q}{\partial t} = 0 \qquad \text{if} \qquad \frac{\partial \eta}{\partial x} = \frac{\partial \eta}{\partial y} = hu = hv = 0, \tag{7}$$

in which η is the sea-surface deviation from a mean equilibrium level, given by

$$\eta = h - H. \tag{8}$$

Kurganov and Levy [11] presented a discretization of the bottom slope source term $B(Q)$ and a matching reconstruction based on water elevation to capture such steady states. For oceanographic simulation, an important steady state is the so-called geostrophic balance, in which the angular momentum caused by the rotational reference frame is balanced by the pressure gradient:

$$fhv - gh\frac{\partial \eta}{\partial x} = 0, \qquad -fhu - gh\frac{\partial \eta}{\partial y} = 0. \tag{9}$$

The novel idea in the CDKLM scheme is to reconstruct the face values $Q^l_{i+1/2}$ and $Q^r_{i+1/2}$ based on this geostrophic balance so that the resulting flux $F_{i+1/2}$ also balances the Coriolis force for steady-state solutions. This makes the scheme suitable for simulating ocean currents in a rotating frame of reference.

3 Efficient simulation of real-world ocean currents

Our simulator is implemented on the GPU for efficiency. Explicit finite-volume schemes normally have an embarrassingly parallel nature, and it is therefore natural to implement them on such massively parallel accelerator hardware

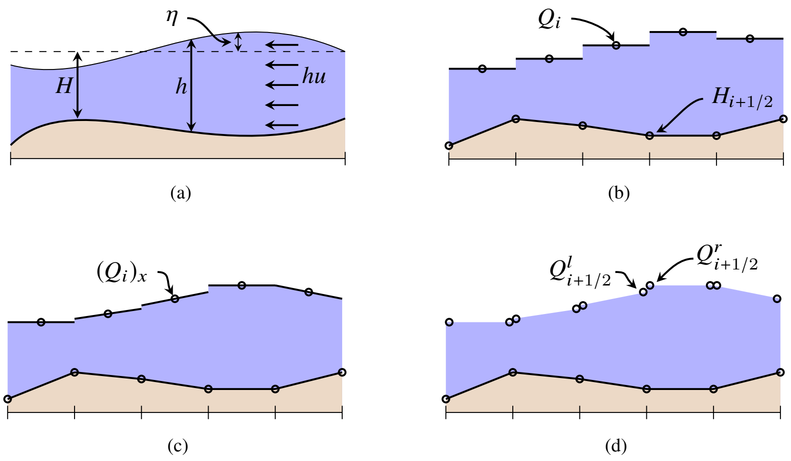


Figure 1: a) The relationship between the physical variables h, H, η, and hu used in the shallow-water equations, here shown in one dimension. b) The ocean state is discretized in terms of cell average values, whereas the bathymetry is defined at the cell intersections. c) We find the slopes of the ocean state within each cell. d) The flux between the cells is found from two one-sided point values at each interface, as seen from the two adjacent cells.

(see e.g., [12, 13, 14, 15]). We use CUDA [16] coupled with PyCUDA [17] to access the GPU from Python in this work, and we have previously shown that using Python instead of C++ has a negligible performance impact when used like this [18]. Using the Python ecosystem for tasks such as pre-processing input data and post-processing results has significantly increased our productivity, while still maintaining high performance.

A simulation starts by reading initial conditions, forcing terms, and other grid data from NetCDF files hosted by *thredds* servers at the Norwegian Meteorological Institute. We continue by initializing the simulator, which internally allocates memory and uploads initial conditions on the GPU. Finally, the main loop steps the simulator forward in time, and occasionally (e.g., every simulated hour) downloads and writes results to NetCDF files on disk.

The rest of this section covers additions and improvements to the CDKLM scheme required for simulating real-world scenarios.

3.1 Efficient parallel formulation of CDKLM

The original CDKLM scheme uses a recursive formulation of the potential energies used for the well-balanced flux reconstruction. This turns the reconstruction procedure into a global operation with a data-dependency spanning the whole domain, making the scheme prohibitively expensive on parallel architectures. The recursive expression can be calculated somewhat efficiently using a prefix sum, as suggested by the authors themselves, but by carefully reformulating the recursive terms it can also be rephrased as a local operation.

The scheme reconstructs h, hu, and hv on each side of a face to calculate the fluxes in (6). However, to capture the rotating steady-state solutions in geostrophic balance, the reconstruction is based on the potential energies,

$$K = g(\eta - V), \qquad L = g(\eta + U), \tag{10}$$

instead of physical variables. Here, U and V are the primitives of the Coriolis force, given by

$$V_x = \frac{f}{g}v, \qquad U_y = \frac{f}{g}u. \tag{11}$$

Note that setting $K_x = 0$ and $L_y = 0$ is equivalent to the geostrophic balance from (9). In the following, we detail the reconstruction along the abscissa, but the same derivations apply in the ordinate. The recursive terms appear from the discretization of U and V, which are defined on the cell faces as

$$V_{i+1/2,j} = V_{i-1/2,j} + \frac{\Delta x}{g} f_{i,j} v_{i,j}, \qquad V_{-1/2,j} = 0. \tag{12}$$

The Coriolis parameter $f_{k,j}$ is allowed to change both along the x and y axis, even though f only varies with latitude. This is because we will use a spatially varying latitude, as described in more detail in Section 3.7. Values of V in the cell centers are obtained by taking the mean of the face values, so that the cell-average value of K from (10) becomes

$$K_{i,j} = g\left(\eta_{i,j} - \frac{1}{2}\left(V_{i+1/2,j} + V_{i-1/2,j}\right)\right). \tag{13}$$

By reconstructing the (limited) slopes of K within each cell, the face values of h are found by combining (13) and (8), so that

$$h^E_{i,j} = \frac{1}{g}\left(K_{i,j} + \frac{\Delta x}{2}(K_x)_{i,j}\right) + V_{i+1/2,j} + H_{i+1/2,j}, \tag{14}$$

and

$$h^W_{i,j} = \frac{1}{g}\left(K_{i,j} - \frac{\Delta x}{2}(K_x)_{i,j}\right) + V_{i-1/2,j} + H_{i-1/2,j}. \tag{15}$$

The reconstructed values (14) and (15) of h at the cell faces quickly become a performance bottleneck as long as they depend on calculating the recursive relation in (12). As it turns out, however, we never need to explicitly compute the recursive terms. First of all, notice that (14) and (15) only depend on derivatives of K and not on L. Similarly, the reconstructions of $h^N_{i,j}$ and $h^S_{i,j}$ depend on L and L_y. This means that we are only interested in derivatives of K (and L) in the same direction as the recursive terms for V (and U). The derivatives are limited using the generalized minmod function using the backward, central, and forward differences. The backward difference is given by

$$\frac{K_{i,j} - K_{i-1,j}}{\Delta x} = \frac{g}{\Delta x}\left(\eta_{i,j} - \frac{1}{2}\left(V_{i+1/2,j} + V_{i-1/2,j}\right) - \eta_{i-1,j} + \frac{1}{2}\left(V_{i-1/2,j} + V_{i-3/2,j}\right)\right). \tag{16}$$

We see that $V_{i-1/2,j}$ cancels, and focus on the remaining V values. By applying the recursive expression in (12) twice on $V_{i+1/2,j}$ we get

$$\begin{aligned}\frac{1}{2}\left(-V_{i+1/2,j} + V_{i-3/2,j}\right) =& \frac{1}{2}\left(-V_{i-1/2,j} - \frac{\Delta x}{g} f_{i,j} v_{i,j} + V_{i-3/2,j}\right)\\ =& \frac{1}{2}\left(-V_{i-3/2,j} - \frac{\Delta x}{g} f_{i-1,j} v_{i-1,j} - \frac{\Delta x}{g} f_{i,j} v_{i,j} + V_{i-3/2,j}\right)\\ =& -\frac{\Delta x}{2g}\Big(f_{i-1,j} v_{i-1,j} + f_{i,j} v_{i,j}\Big).\end{aligned} \tag{17}$$

Inserted back into (16), the backward difference needed to evaluate $(K_{i,j})_x$ can be written as

$$\frac{K_{i,j} - K_{i-1,j}}{\Delta x} = \frac{g}{\Delta x}\left(\eta_{i,j} - \eta_{i-1,j} - \frac{\Delta x}{2g}\Big(f_{i-1,j} v_{i-1,j} + f_{i,j} v_{i,j}\Big)\right), \tag{18}$$

which no longer contains any recursive terms. Similar derivations can be used for the forward and central differences, and equivalently to obtain $(L_{i,j})_y$.

After obtaining $(K_x)_{i,j}$ we look at the reconstruction of $h^E_{i,j}$ from (14). We start by inserting the expression for K from (13) into (14) and gather all V-terms,

$$\begin{aligned}h^E_{i,j} &= \eta_{i,j} - \frac{1}{2}\left(V_{i+1/2,j} + V_{i-1/2,j}\right) + \frac{\Delta x}{2g}(K_x)_{i,j} + V_{i+1/2,j} + H_{i+1/2,j}\\ &= \eta_{i,j} + H_{i+1/2,j} + \frac{\Delta x}{2g}(K_x)_{i,j} + \frac{1}{2}\left(V_{i+1/2,j} - V_{i-1/2,j}\right).\end{aligned} \tag{19}$$

Here, the values for $\eta_{i,j}$, $H_{i,j}$ and $(K_x)_{i,j}$ are known, and by using the recursive expression in (12) we get that

$$\begin{aligned}\frac{1}{2}\left(V_{i+1/2,j} - V_{i-1/2,j}\right) &= \frac{1}{2}\left(V_{i-1/2,j} + \frac{\Delta x}{g} v_{i,j} f_{i,j} - V_{i-1/2,j}\right) \\ &= \frac{\Delta x}{2g} f_{i,j} v_{i,j}.\end{aligned} \tag{20}$$

Inserting (20) back into (19) we see that we get

$$h^E_{i,j} = \eta_{i,j} + H_{i+1/2,j} + \frac{\Delta x}{2g}\Big[(K_x)_{i,j} + f_{i,j} v_{i,j}\Big], \tag{21}$$

Again, the same derivation can be applied on the other three face values as well. This enables us to express the reconstruction step in terms of only local variables, which unlocks an embarrassingly parallel numerical algorithm.

3.2 Accuracy, precision and GPUs

Today's GPUs can be categorized in two major classes: GPUs meant for the consumer market, and GPUs meant for supercomputers and other professional users. For the most part, these GPUs are close to identical, but there are some key differences. One is price: professional GPUs can cost over ten times more than equivalent consumer market GPUs, making the latter tractable from a financial stand point. Another difference is in the feature set. Professional GPUs offer more memory, error-correcting code (ECC) memory, and significantly higher performance for features such as double precision.

Double precision is not an important feature on consumer market GPUs, and typically offer just 3% of the performance of single-precision arithmetics. The NVIDIA TITAN RTX, for example, has a performance of 16.3 teraFLOPS in single precision, and 0.51 teraFLOPS in double precision. On professional GPUs, on the other hand, the performance is 50% of single precision, which is the same ratio as on CPUs. If we are able to utilize single-precision calculations, we should therefore be able to get the highest performance at minimum monetary cost. We have previously shown that using single precision is sufficiently accurate for most simulation scenarios for the shallow-water equations [19], and recent studies have shown a 40% increase in performance for complex forecasting models when using single precision [20].

It is well known that floating-point numbers have a round-off error that increases for larger numbers. A single-precision floating-point number is represented as

$$(-1)^s \cdot 2^{e-127} \cdot (1.0 + f) \tag{22}$$

in which s is the sign bit, e is represented using 8 bits, and f is the fractional part, represented by 23 bits. This formulation means that the distance between two floating-point numbers is smallest close to zero, and increases as e increases. This is important to note in oceanographic simulations, as the water depth, h, can often be thousands of meters, whereas the relevant tidal wave height is perhaps 0.5 meters. The loss in precision due to the water depth makes floating point prohibitively inaccurate if we implement the shallow-water equations as formulated in (1). To counter this loss in precision, we can possibly use double precision (at twice the computational and memory cost), but a better alternative is to reformulate the problem to represent the relevant physical quantities. This can be done by basing our simulation on η from (8) instead of h and use $[\eta, hu, hv]^T$ as our vector of conserved variables. By combining this approach with single-precision floating-point numbers, we get more than sufficient accuracy for the relevant oceanographic simulations.

When the water depth becomes close to zero, the calculation of the particle velocity, $u = hu/h$ can become severely ill conditioned as the expression is prone to large round-off errors. To counter this, Kurganov [21] and Kurganov and

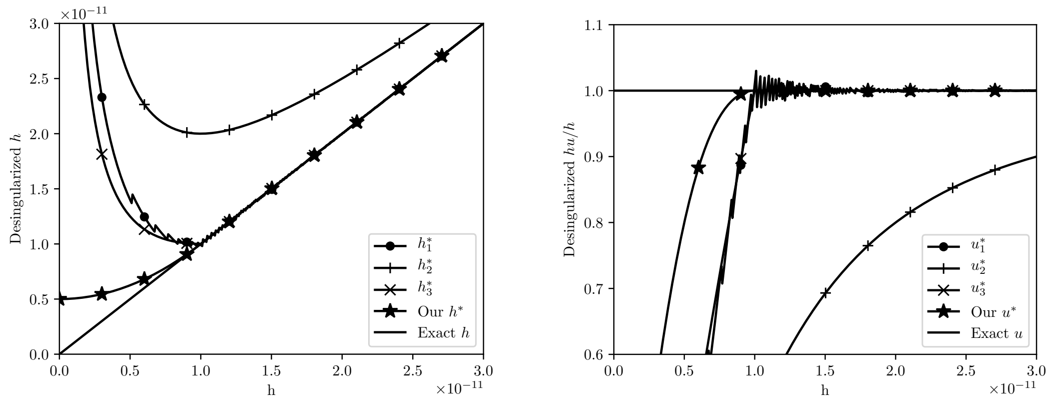


Figure 2: Desingularization of the quantity $u = h/hu$ using existing and our proposed method with $\kappa = 1.0 \cdot 10^{-11}$ for all cases. We compute the value of hu/h using the different approaches for $u = 1$ and different values of h. Notice how the single precision floating-point errors are clearly visible for u_1^*, that u_2^* has a distinct kink, and that u_3^* significantly underestimates the true magnitude of u.

Petrova [22] have proposed to *desingularize* the term using expressions such as

$$u_1^* = \frac{\sqrt{2}h(hu)}{\sqrt{h^4 + \max(h^4, \kappa^4)}} \tag{23}$$

$$u_2^* = \frac{h \cdot hu}{h^2 + \kappa^2}, \qquad \text{or} \tag{24}$$

$$u_3^* = \frac{2h \cdot hu}{h^2 + \max(h^2, \kappa^2)}. \tag{25}$$

The suggestion of using $\kappa = \max(\Delta x^4, \Delta y^4)$ has some obvious problems for real-world simulations with $\Delta x > 1$, as pointed out in [13]. Furthermore, the desingularization of u here is not very well-behaved. Instead, we propose to use the following formulation

$$\begin{aligned} h^* &= \operatorname{sign}(h) \max\Big(|h|, \min(h^2/(2\kappa) + \kappa/2, \kappa)\Big) \\ u^* &= hu/h^*. \end{aligned} \tag{26}$$

The advantages of this formulation is that (i) κ now controls directly what magnitudes of h to desingularize, (ii) it improves the numerical stability as it avoids the square root and the problematic h^4 term, (iii) it yields a smooth transition between normal and desingularized values, and (iv) the effective magnitude of h^* is controlled. Figure 2 shows the three desingularization strategies and clearly indicates that the desingularized quantity h is better behaved with our proposed approach.

3.3 Global wall boundary conditions

We have implemented wall and periodic boundary conditions to be able to run test cases to assess the numerical correctness of our simulator. For reflective von Neumann-type wall boundary conditions, we use so-called ghost cells that mirror the ocean state across the boundary to enforce a zero flux out of the domain. In the absence of Coriolis forces, wall boundary conditions at $x = 0$ are constructed by setting the ghost cell values according to

$$\eta(-x, y) = \eta(x, y), \qquad hu(-x, y) = -hu(x, y), \qquad hv(-x, y) = hv(x, y). \tag{27}$$

Naïve implementation of these conditions with the CDKLM scheme leads to gravitational waves along the boundary. Special care is therefore required to balance the Coriolis potential energies in the reconstruction of h, so that

$$\begin{aligned} 0 &= h^W_{0,j} - h^E_{-1,j} \\ &= \left(\eta_{0,j} + H_{-1/2,j} - \frac{\Delta x}{2g}\Big[(K_x)_{0,j} + f_{0,j}v_{0,j}\Big]\right) - \left(\eta_{-1,j} + H_{-1/2,j} + \frac{\Delta x}{2g}\Big[(K_x)_{-1,j} + f_{-1,j}v_{-1,j}\Big]\right), \end{aligned} \tag{28}$$

using (21). Here, we see that H cancels immediately, whereas η cancels by the use of (27). To make the last terms cancel, we need to set $f_{-1,j}v_{-1,j} = -f_{0,j}v_{0,j}$, which means that in addition to (27), we need to enforce

$$f(-x, y) = -f(x, y). \tag{29}$$

Note also that this results in $(K_x) = 0$ across the boundary from (18).

3.4 Moving wet-dry boundary and land mask

The original CDKLM scheme breaks down on land as the gravitational wave speed $\sqrt{gh}$ becomes imaginary with negative depths. To support a moving wet-dry boundary, we use a similar approach as Kurganov and Petrova [22] by adjusting the slope of h so that all the reconstructed point values become non-negative. For example, if we get $h^E_{i,j} < 0$ from (21), we adjust the slope to become

$$(K_x)_{i,j} = -f_{i,j}v_{i,j} - \frac{2g}{\Delta x}\Big[\eta_{i,j} + H_{i+1/2,j}\Big], \tag{30}$$

and thus forcing $h^E_{i,j} = 0$ instead.

The moving wet-dry boundary is required for simulating, e.g., tsunamis and other phenomena in which the run-up is important. The approach nonetheless violates the steady-state reconstruction, causing non-physical waves and often also small time step sizes. For static wet-dry boundaries, we have therefore implemented wall boundaries following a land mask, i.e., ensuring a zero flux for the shore-line. In our approach, we explicitly set the flux between two neighboring cells to zero if any one of them is masked. The mask is represented by a unique no-data value in the bathymetry, thereby having no increase in the memory footprint. It should be noted that both of these approaches reduce the reconstruction to first order along the wet-dry boundary.

3.5 Wind forcing and bed friction source terms

The original formulation of CDKLM includes the bed slope source term, $B(Q)$, and Coriolis force, $C(Q)$, and we extend the scheme to also include the bed shear stress, $S(Q)$, and wind forcing, $W(Q)$. Bed shear stress is discretized using a semi-implicit formulation in the strong stability preserving Runge-Kutta scheme,

$$\begin{aligned} Q^*_{i,j} &= \left(Q^n_{i,j} + \Delta t M(Q^n)_{i,j}\right) / \left(1 + \Delta t S(Q^n_{i,j})\right) \\ Q^{n+1}_{i,j} &= \tfrac{1}{2}\left(Q^n_{i,j} + \left(Q^*_{i,j} + \Delta t M(Q^*)_{i,j}\right)\right) / \left(1 + \tfrac{1}{2}\Delta t S(Q^*_{i,j})\right), \end{aligned} \tag{31}$$

instead of (4). This essentially means that we apply half of the bed friction using Q^n, and half of the friction using the predictive step Q^*.

The wind stress comes from the atmospheric wind transferring momentum to the water column. We take the approach of Large and Pond [23] and use

$$\vec{\tau} = \frac{\rho_a}{\rho_w} C_D \left|W_{10}\right| W_{10}, \tag{32}$$

in which $\rho_a = 1.225$ is the specific density of air, $\rho_w = 1025$ is the specific density of sea water, W_{10} is the wind speed at 10 meters, and C_D is the drag coefficient computed as

$$C_D = 10^{-3}\begin{cases} 1.2 & \text{if} \quad |W_{10}| < 11 \text{ m/s}, \\ 0.49 + 0.065\,|W_{10}| & \text{if} \quad |W_{10}| \geq 11 \text{ m/s}. \end{cases} \tag{33}$$

The NorKyst-800 model stores its output, including the wind forcing, every hour. Because the time step of our simulator is significantly smaller than one hour, we use linear interpolation in time to get the most accurate wind stress source term for each time step. Given the wind stresses W_{10}^a at time t_a and W_{10}^b at time t_b, we compute the wind stress at time $t \in [t_a, t_b]$ as a convex combination of the two

$$W_{10}^n = (1-s)W_{10}^a + sW_{10}^b, \qquad s = (t-t_a)/(t_b-t_a). \tag{34}$$

In space, we use specialized GPU texture hardware for bilinear interpolation (see e.g., [24]), resulting in a trilinearly interpolated wind stress. This is highly efficient, as the GPU has dedicated hardware for bilinear spatial interpolation that also includes caching. An added benefit is that the grid size of the wind stress data does not need to match up with the underlying simulation grid.

3.6 Nesting the model into NorKyst-800

A common technique in operational oceanography is to nest high-resolution local models within lower-resolution models that cover a larger domain. For instance, the NorKyst-800 model is nested within TOPAZ, which is a 12-16 km resolution HYCOM-based coupled ocean and sea ice model with data assimilation [25]. In this work, we nest our simulations into the NorKyst-800 model by setting initial ocean state and boundary conditions appropriately. We use the so-called flow relaxation scheme [26], in which we have a relaxation zone between the internal computational domain and the external boundary conditions. In this region, we solve the shallow-water equations as normal to obtain an internal solution $Q_{i,j}^{int}$, and then gradually nudge it towards an external solution Q^{ext} using

$$Q_{i,j} = (1-\alpha)Q_{i,j}^{int} + \alpha Q^{ext}, \qquad \alpha = 1 - \tanh(d_{i,j}/d_0). \tag{35}$$

Here, $d_{i,j}$ is distance in number of cells to the external boundary, and the parameter d_0 is typically set to 2 or 3 to control the fall-off of the $\tanh$ relaxation function.

The implementation of the boundary conditions follows the same ideas as for the wind stress, using textures for linear interpolation in time and space. We use float4[2] textures to hold our physical variables, η, hu, hv, because float3 is not supported as a texture format. We furthermore only use two textures, as we pack the north and south boundaries into one texture, and similarly for the east and west.

3.7 Coriolis force

The original CDKLM scheme requires that the grid is aligned with the cardinal directions, which significantly restricts the use of the scheme. For higher latitudes this becomes especially pronounced, as both the latitude and direction towards north will vary across the domain. We have therefore extended the scheme to support both a spatially varying north-vector and a spatially varying latitude, which is required to be able to run realistic simulations for the areas covered by the NorKyst-800. To account for the varying north vector, we project the momentum onto the local north and east vectors,

$$\begin{aligned} hu^e &= \vec{e} \cdot [hu, hv]^T, \qquad & \vec{e} &= [\cos(\theta), -\sin(\theta)] \\ hv^n &= \vec{n} \cdot [hu, hv]^T, \qquad & \vec{n} &= [\sin(\theta), \cos(\theta)], \end{aligned} \tag{36}$$

and use these vectors when computing the source term $C(Q)$ so that

$$C(Q) = f \cdot \begin{bmatrix} 0 \\ \vec{x} \cdot [hv^n, -hu^e]^T \\ \vec{y} \cdot [hv^n, -hu^e]^T \end{bmatrix}. \tag{37}$$

in which the vectors $\vec{x} = [\cos(\theta), \sin(\theta)]$ and $\vec{y} = [-\sin(\theta), \cos(\theta)]$ project the result back into the (x, y)-coordinate system. Note that we also need to use the same approach in the reconstruction of h as discussed in Section 3.1 to maintain the rotational steady states.

[2]Textures on GPUs are originally designed to hold the color of one pixel as the color channels red, green, blue, and alpha, hence float4.

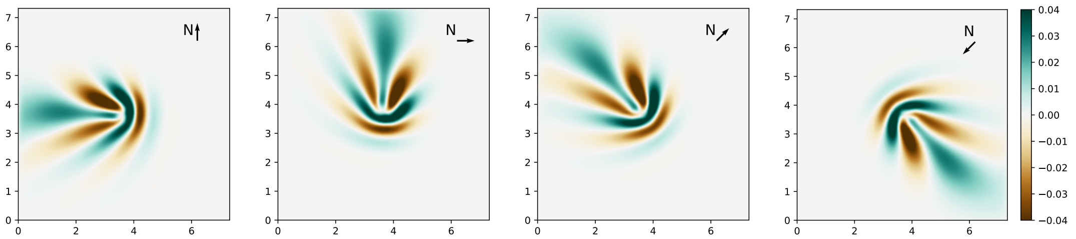


Figure 3: Planetary Rossby waves generated by a beta plane model for the Coriolis force and with different directions for north. The simulations are initialized by a rotating bump in the middle of the domain, which develops into Rossby waves that propagates westwards with wave energy slowly propagating eastwards. The different figures show that we get the same Rossby waves for arbitrary orientation of our domain. Axes are given in 1000 km, and the colormap shows η in m.

Our spatially varying Coriolis parameter is computed from the latitude, $l_{i,j}$, of a cell

$$f_{i,j} = 2\omega \sin(l_{i,j}), \qquad \omega = 7.2921 \cdot 10^{-5}\text{rad/s}, \tag{38}$$

or using a beta-plane model,

$$f_{i,j} = f_{ref} + \beta \cdot \left(\vec{n} \cdot \left[(x_i - x_{ref}), (y_j - y_{ref})\right]^T\right). \tag{39}$$

Here, f is linearized around a reference point, (x_{ref}, y_{ref}), with a slope β in the direction of $\vec{n}$. Setting β to zero yields a constant Coriolis parameter throughout the domain.

Figure 3 demonstrates how the angle to north plays an important role in the generation of planetary Rossby waves, caused by variations in the Coriolis force using (39). The figure shows the resulting η after long simulations initialized with a rotating Gaussian bump in geostrophic balance in the middle of the domain, similar to the case presented in Holm et al. [27]. Here, we have used a domain consisting of 350×350 cells with $\Delta x = \Delta y = 20$ km, a depth of $H = 50$ m, and the center of the domain corresponding to approximately 33° north, with $f = 8 \cdot 10^{-4}$ and $\beta = 2 \cdot 10^{-11}$. The simulation ends after approximately 35 days. The figure clearly shows the effect of varying the north vector and hence the Coriolis source term.

3.8 Bathymetry

The bathymetry in the NorKyst-800 model is given as cell midpoint values, and can not be used directly in our framework as CDKLM requires the bathymetry to be defined as a piecewise bilinear surface specified by point values at the cell intersections. It is still desirable that the averages of the resulting four intersection values end up to be equal to the NorKyst-800 cell average values, as large deviations can disturb the balanced relations in the provided ocean state. Constructing such a piecewise bilinear surface from the cell averages is an ill-posed problem, and a simple averaging of cell values excessively smears important features such as deep fjords and straits.

We have therefore devised a pragmatic iterative algorithm, which is based on reconstruction of slopes for each cell, similar to the reconstruction procedure in our numerical shallow-water scheme. Our initial guess is computed as follows:

1. Compute the local gradient in each cell so that

$$[H^x_{i,j}, H^y_{i,j}] = \nabla H_{i,j}. \tag{40}$$

2. Evaluate the piecewise planar function at the four corners of each cell,

$$\begin{aligned}
H^{i,j}_{i-1/2,j-1/2} &= H_{i,j} - \tfrac{1}{2}H^x_{i,j} - \tfrac{1}{2}H^y_{i,j}, \\
H^{i,j}_{i+1/2,j-1/2} &= H_{i,j} + \tfrac{1}{2}H^x_{i,j} - \tfrac{1}{2}H^y_{i,j}, \\
H^{i,j}_{i-1/2,j+1/2} &= H_{i,j} - \tfrac{1}{2}H^x_{i,j} + \tfrac{1}{2}H^y_{i,j}, \\
H^{i,j}_{i+1/2,j+1/2} &= H_{i,j} + \tfrac{1}{2}H^x_{i,j} + \tfrac{1}{2}H^y_{i,j}.
\end{aligned} \tag{41}$$

3. Average the estimate from the four cells meeting at one intersection point so that

$$H^*_{i+1/2,j+1/2} = \tfrac{1}{4}\left(H^{i,j}_{i+1/2,j+1/2} + H^{i+1,j}_{i-1/2,j+1/2} + H^{i+1,j+1}_{i-1/2,j-1/2} + H^{i,j+1}_{i+1/2,j-1/2}\right) \tag{42}$$

After obtaining the initial guess, we start an iterative procedure consisting of two stages. The first stage minimizes the difference between computed and true midpoint values, and the second dampens oscillations created by the first stage. We start by computing the difference between our estimate and the target bathymetry as

$$\Delta H_{i,j} = H_{i,j} - H^*(i,j), \tag{43}$$

in which $H^*(i,j)$ means that we evaluate the piecewise bilinear estimate at midpoints. We then apply (40)–(42) to compute an estimate of the difference at intersections, $\Delta H^*_{i+1/2,j+1/2}$, and subtract this difference from our initial guess,

$$H^*_{i+1/2,j+1/2} := H^*_{i+1/2,j+1/2} + \Delta H^*_{i+1/2,j+1/2}. \tag{44}$$

This brings our updated midpoints of $H^*(i,j)$ closer to the target $H_{i,j}$, but it also results in unwanted oscillations, as there is nothing that limits the slopes of this piecewise bilinear surface. We therefore smooth H^*, and repeat the procedure until the update between two subsequent iterations is sufficiently small. Around ten iterations is typically sufficient to create a piecewise bilinear surface that lies close to the target. Figure 4 shows the result of our approach applied to the bathymetry from NorKyst-800 in the innermost parts of Vestfjorden (see also Section 5.2).

CDKLM is a full two-dimensional scheme, yet it exhibits a grid orientation bias close to the land mask. For example, we have experienced that the scheme has difficulties in propagating the momentum through narrow straits and canals that are only one or two cells wide. To ameliorate this, we can use binary erosion on the land mask to erode one row of cells at a time as shown in Figure 5. We extrapolate η into these areas using gray dilation (maximum of neighbors), and set the water depth equal the land value, which in the NorKyst-800 model is 5 meters. The process can be repeated to erode the land mask even more, and is required for tidal waves to properly propagate in features such as narrow fjords and straits. We have found that using single iteration is sufficient to propagate tides in and out of the fjords in the NorKyst-800 model.

3.9 Refinement and coarsening of grid

The possibility to run simulations at different resolutions is attractive to quickly obtain low-resolution preliminary results, and also to resolve more of the dynamics for more detailed and accurate results with higher resolution. We have included functionality in our simulation framework to increase and decrease the grid resolution with factors of two to support both of these scenarios. We use a static grid refinement/coarsening performed on the initial conditions themselves, but the same approach can be used for efficient adaptive mesh refinement following the same approach as Sætra, Brodtkorb and Lie [28].

The physical variables are represented by cell average values, and refinement is based on a slope-limited reconstruction within each cell, which we evaluate at the refined grid cell centers. As the bathymetry is specified by a piecewise bilinear surface, the high-resolution representation of the bathymetry evaluates the surface on the refined cell intersections. Low-resolution representation of the physical variables on a coarsened grid is simply the average of the original cell values, and intersection values for the bathymetry are sampled from the corresponding original intersection values. With a factor two coarsening, both these operations are well-defined.

Note that the boundary conditions and wind forcing terms are independent of the mesh resolution due to our use of GPU textures.

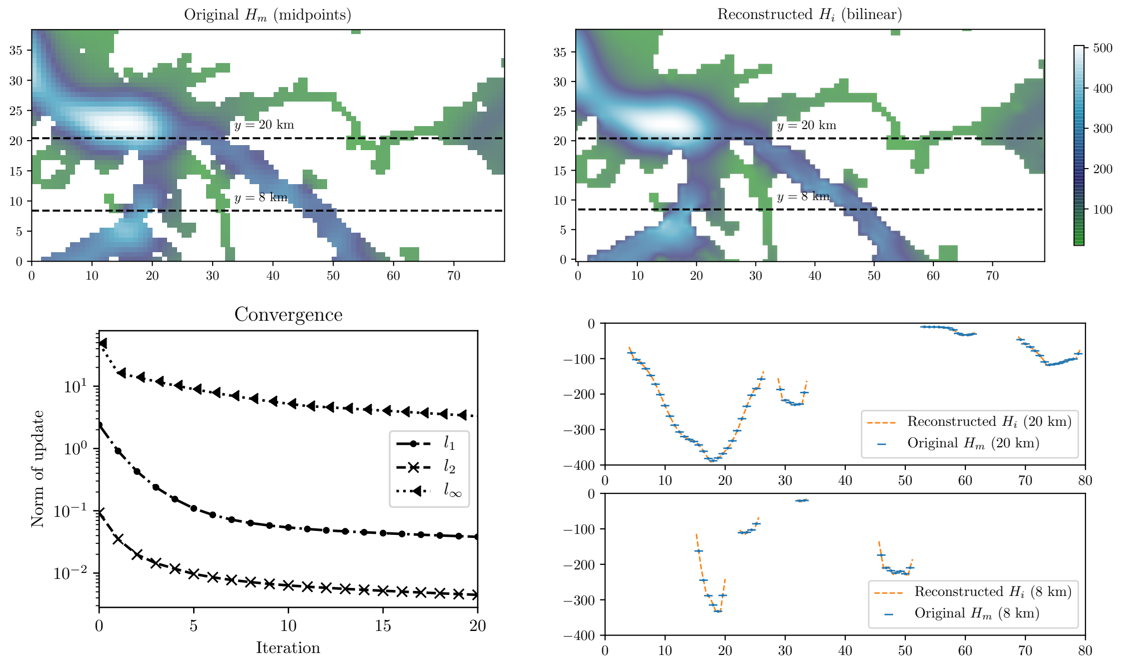


Figure 4: Reconstruction of the bathymetry at intersection points. Starting with the initial bathymetry H_m defined at cell midpoints (upper left), we compute the H_i values at intersections (upper right). The lower-left figure shows the convergence of our reconstruction algorithm, and the reconstructed bathymetry is plotted together with the original in the lower-right figure.

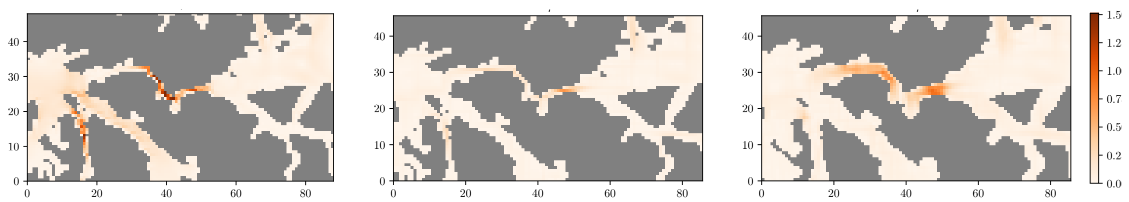


Figure 5: Widening of narrow straits, in which the land mask is eroded using binary erosion. The left figure shows particle velocities in the Tjeldsund strait in Lofoten/Vesterålen between the main land and Hinnøya (the bathymetry is shown in Figure 4). Left shows the NorKyst-800 reference, center shows original (reconstructed) bathymetry, and the right figure shows the result with one grid cell land erosion. The strait width and geometry limits the amount of water passing through it in the original bathymetry, but by eroding one cell of the land mask the amount of water is much closer to the reference.

Table 1: Error of computed solutions at different resolutions and numerical reduction order for each refinement. The time step is fixed throughout these simulations.

Resolut	Δt	L_1	**Order**	L_2	**Order**	L_{inf}	**Order**
16^2	1600/16	0.002898	-	0.006072	-	0.037367	-
32^2	1600/32	0.000923	1.65	0.001877	1.69	0.012651	1.56
64^2	1600/64	0.000256	1.85	0.000502	1.90	0.002621	2.27
128^2	1600/128	0.000058	2.14	0.000122	2.04	0.000741	1.82
256^2	1600/256	0.000015	1.96	0.000033	1.87	0.000322	1.20
512^2	1600/512	0.000005	1.58	0.000009	1.89	0.000136	1.25

4 Performance and accuracy of GPU implementation

We have previously evaluated the standard CDKLM scheme on several test cases which target oceanographic dynamics relevant for the shallow-water equations when implemented on a Cartesian grid [27]. In this work, however, we have reformulated the scheme in terms of η instead of h, added several new source terms, and enabled varying latitude and north vector. It is therefore important to revisit the grid convergence test. We also measure computational performance of our simulation framework through assessing its weak scaling.

4.1 Convergence of the numerical scheme

To evaluate numerical convergence, we define a benchmark case covering $500\text{ km} \times 500\text{ km}$. We model the bathymetry after the Matlab `peaks` function,

$$\text{peaks}(s,t) = 3(1-s)^2 e^{-s^2-(t+1)^2} - 10(s/5 - s^3 - t^5)e^{-s^2-t^2} - 1/3e^{-(s+1)^2-t^2}, \qquad s,t \in [-3,3] \tag{45}$$

which yields a non-symmetric smooth surface. We then define the equilibrium ocean depth as

$$H(x,y) = 100 + 10 \cdot \text{peaks}\left(\frac{6x}{500} - 3, \frac{6y}{500} - 3\right), \tag{46}$$

with x and y given in km, so that the depth varies between 34 and 181 meters (see Figure 6). We initialize the sea-surface level with a bump centered in the domain,

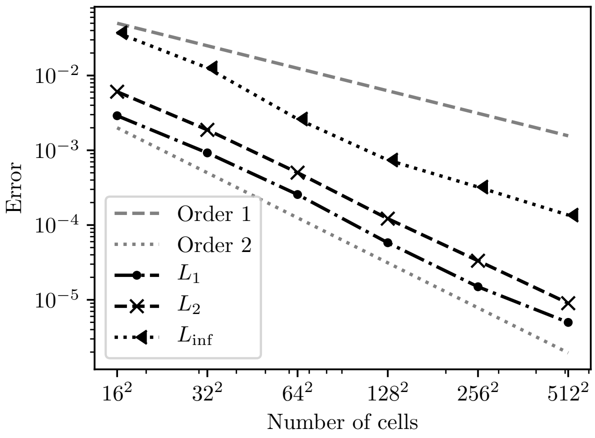
$$\eta(r) = \begin{cases} \frac{1}{2}\left(1 + \cos\left(\frac{r}{c}\pi\right)\right), & \text{if } r \le c, \\ 0, & \text{if } r > c, \end{cases} \tag{47}$$

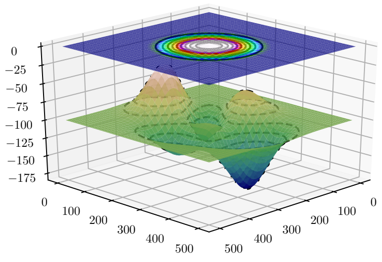
in which $r = \sqrt{(x-250)^2 + (y-250)^2}$ is the distance from the center of the domain, and c controls the size of the bump, set to 300 km in our case. We assume that the North pole is located at $(3000\text{ km}, 3000\text{ km})$, which means that the angle between the y-axis and the north vector varies from $33.7°$ to $56.3°$ across the domain and that the latitude varies from $56.0°$ to $67.4°$ north. These parameters roughly correspond to the North Sea. Including land inevitably yield only first-order accurate fluxes, and we have thus deliberately not included such areas in our grid convergence test.

The simulation is run for 800 seconds, after which we compute the difference for the whole domain between the solution and a reference computed on a 1024×1024 grid. Figure 6 and Table 1 show that we achieve second-order accuracy in both the L_1 and the L_2 norm, whilst the convergence in L_{inf} is somewhat slower.

4.2 Computational performance

To assess the computational performance, we continue to use the same benchmark, but run the simulation twice as long. We also measure the cost of downloading the results from the GPU to the CPU. Figure 7 shows the results for





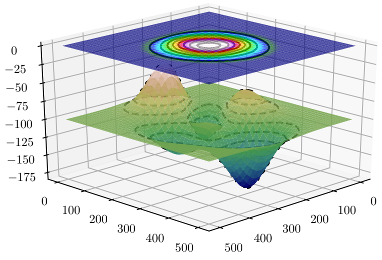


Figure 6: Convergence plot of our numerical scheme. The numerical case has a water disturbance consisting of a cosine bump at the center of the domain, and the bathymetry consists of the smooth Matlab peaks function. The Coriolis force varies spatially across the domain according to variations in latitude and direction towards north. The left figure shows the self convergence of the numerical error compared to the reference solution with 1024^2 cells. To the right we show the bathymetry with the sea-surface level initially (top) and at the end of the simulation (bottom).

Table 2: Performance for our simulator. The simulation time is approximately eight times larger when we double the resolution. This is because we have four times as many cells and twice as many timesteps to reach the same end time. The download increases by a factor four, which is to be expected as we transfer four times as many data elements.

Resolution	**Simulation time**	**Factor**	**Download time**	**Factor**
512^2	0.169710	-	0.000929	-
1024^2	1.170814	6.9	0.002773	3.0
2048^2	9.133564	7.8	0.019642	7.1
4096^2	72.832028	8.0	0.077541	3.9
8192^2	582.599290	8.0	0.305405	3.9
16384^2	4695.904546	8.0	1.218880	4.0

domains with 16^2 to 16 384^2 cells, running on a Tesla P100 GPU. We see that for the larger domain sizes, we obtain perfect weak scaling, as a factor two grid refinement leads to a factor eight increase in computational complexity (the timestep size is cut in half due to the CFL condition, and we have four times the number of cells). Cases with less than 512^2 cells are dominated by overheads and hence, the cost of a single timestep is close to constant. Time for downloading data scales linearly with the problem size, as is expected. Again, we see the same behavior for small domains as for the computational performance.

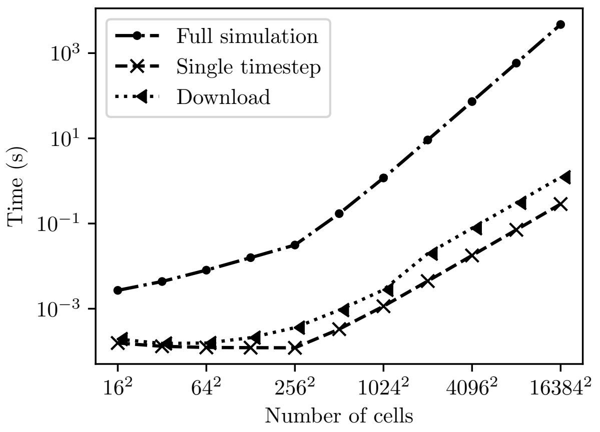


Figure 7: Weak scaling of our simulator. For small domain sizes, the runtime is dominated by overheads, but for domain sizes larger than approximately 512×512, we see that the computational time scales with the number of cells. For each doubling of our domain, we perform approximately eight times as many operations (four times as many cells, and twice the number of time steps).

5 Real-world simulations

Using the NorKyst-800 model system, the Norwegian Meteorological Institute issues daily ocean forecasts for the complete Norwegian coast and makes them publicly available through thredds servers. The results contain full 3D values as well as depth-averaged values, wind forcing, bathymetry, longitudes, latitudes, and angle towards north for each cell. This means that it is simply a matter of accessing the file and cutting out the correct sections to be able to initialize our simulator. To start a simulation within our framework, we simply need the full url to the specific forecast we want to nest our simulation within, and specify a desirable sub-domain.

We run our simulations on the three different sub-domains shown in Figure 8. Our first simulation is in the Norwegian Sea with no land values within the domain, and is therefore a fairly simple test for using real initial and boundary conditions. The second case is centered on the Lofoten archipelago in Northern Norway, which is dominated by a complex coastline consisting of narrow fjords and a large number of islands, and serves as a challenging test for our reconstruction of the bathymetry and land mask. Finally, we run almost the complete domain, leaving only 25 cells in each direction to serve as boundary conditions. With the entire Norwegian coastline, it becomes important to account for the difference in the north vector throughout the domain (see the longitudinal lines and north arrows in Figure 8), and we should see the tides following the coast as Kelvin waves. The last case is also used to make sea-level predictions at five different locations, indicated as stars in Figure 8.

All three cases are run with three different resolutions; a high resolution with a grid refined to 400 m cells, the original 800 m resolution of NorKyst-800, and a low resolution grid consisting of 1600 m cells. We use an adaptive time step, and update the time step every 20 minutes according to the CFL condition in (5) with a Courant number 0.8. Each dataset provided from the NorKyst-800 forecasts covers 23 hours, which means that we only have boundary conditions to run simulations for the same time range.

5.1 Case 1: Norwegian Sea

The first case runs an open water domain in the Norwegian Sea and tests our ability to use the NorKyst data as initial and boundary conditions. The sea-surface level changes significantly during the simulation time range as the tidal waves enter and exit the domain, and this dynamic is largely dependent on the boundary conditions. The velocity field on the other hand, is dominated by already present features of slowly rotating dynamics, and will not change much

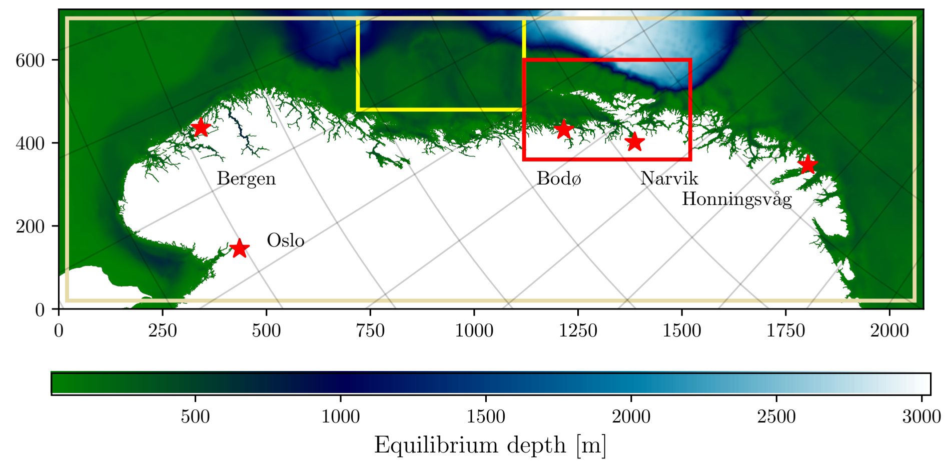


Figure 8: The domain covered by the NorKyst-800 model showing the equilibrium depth off shore. The location of Case 1 in the Norwegian Sea (yellow rectangle), Case 2 around the Lofoten archipelago (red rectangle), and Case 3 (beige rectangle) within the NorKyst-800 model domain, with red stars marking the locations for sea-level predictions. The values on the x- and y-axes are given in km, and the latitude-longitude grid shows how the direction towards north changes throughout the domain.

during the simulation time range. This enables us to test both that the initial conditions are correctly captured, and how well our simulator maintains complex rotating structures over time.

Figure 9 shows simulation results of the Norwegian Sea for all three different grid resolutions after 23 hours, compared to the reference solution from NorKyst-800. First, we see that the contour lines for the sea-surface level in the left column show the same values at similar places for all simulations. This shows that the boundary conditions allow tidal waves to propagate correctly in and out of the domain as intended. The right column shows the corresponding particle velocities. Note that the initial state for both the high-resolution and original runs are visually inseparable from the NorKyst-800 model state, whereas the low-resolution initial conditions slightly smear out the sharper features for the velocity. At 23 hours, we see that the high-resolution model to a large degree maintains the sharp features from NorKyst-800. With the original resolution, however, the particle velocities resemble a slightly blurred image of the reference solution, and in the low-resolution simulation, the features are blurred even more. Also, we are starting to see clear signs of grid effects both internally and from the boundary.

5.2 Case 2: Lofoten

The second case is centered around the Lofoten archipelago in Northern Norway. This region has a complex coast line consisting of a large number of fjords, straits and islands. In addition to the aspects tested in the first case, this case tests the reconstruction of the bathymetry at intersections and the handling of dry zones through the complex land mask. We expect to see a Kelvin wave following the coast, but because the Lofoten archipelago protrudes into the Atlantic, some of the wave will be partially trapped and amplified in Vestfjorden between Lofoten and the main land. Furthermore, as can be seen from the depth map in Figure 8, the edge of the continental shelf crosses through the upper right corner of the sub-domain, and we can here see a line of eddies driven by baroclinic instabilities.

We again run simulations with the three different grid resolutions, and Figure 10 shows the final simulation state after 23 hours, with a high tide in Vestfjorden. All three simulations show that the contour lines for the sea-surface level are slightly off compared to the reference solution, with marginally more water into the fjord. Similarly as in Case 1, the low-resolution simulation is unable to preserve the fine structures in the currents and shows grid effects. The original and high grid resolutions are also unable to preserve the exact eddy structure as found in the reference

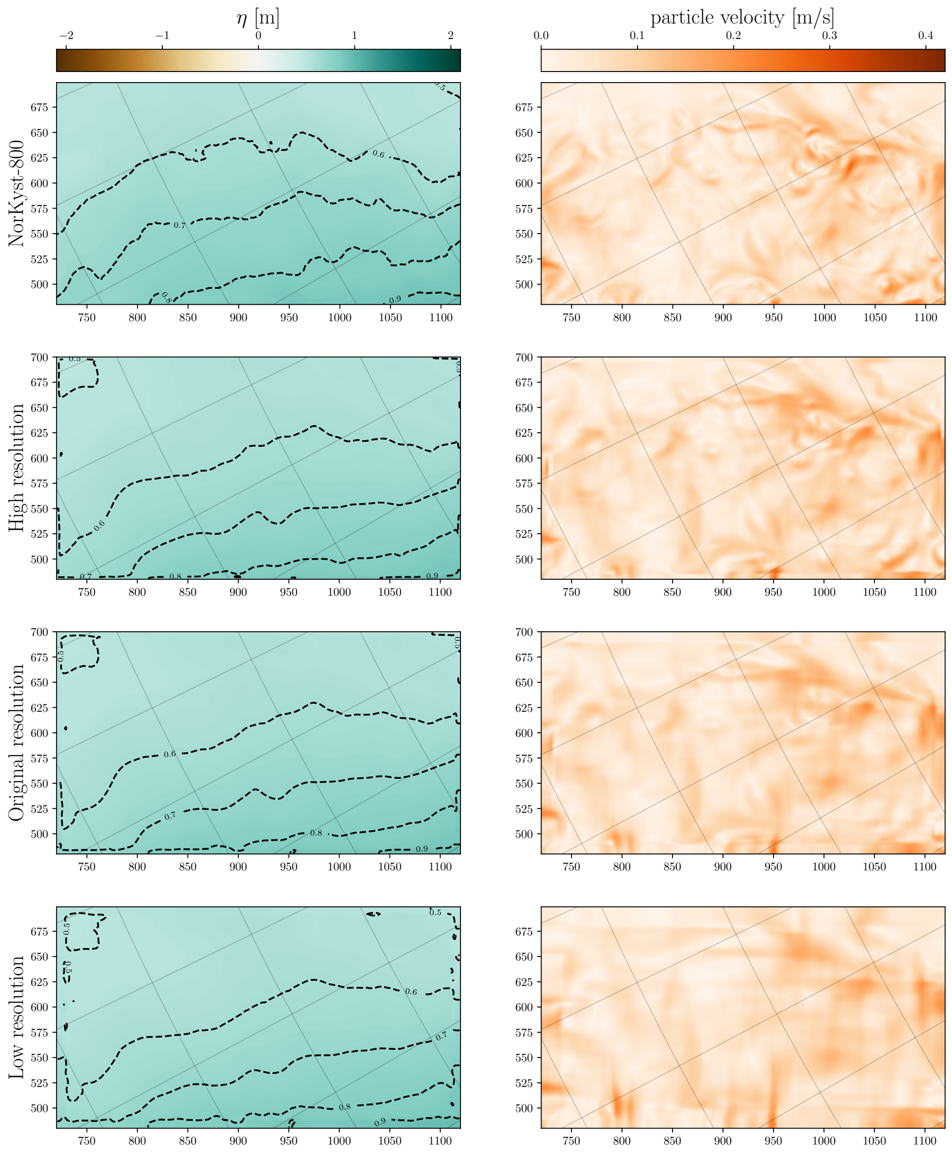


Figure 9: Case 1 of the Norwegian Sea after 23 hours, comparing the reference solution from NorKyst-800 with our simulations using three different grid resolutions. All simulations obtain sea-surface level (left) similar to the reference solution, meaning that the boundary conditions allow the tides to correctly enter and exit our domain. From the particle velocities (right), we see that only the high-resolution model is able to maintain sharp features, whereas the low-resolution simulation has strong signs of grid effects. The values on the x- and y-axes are in km relative to the location in the complete domain.

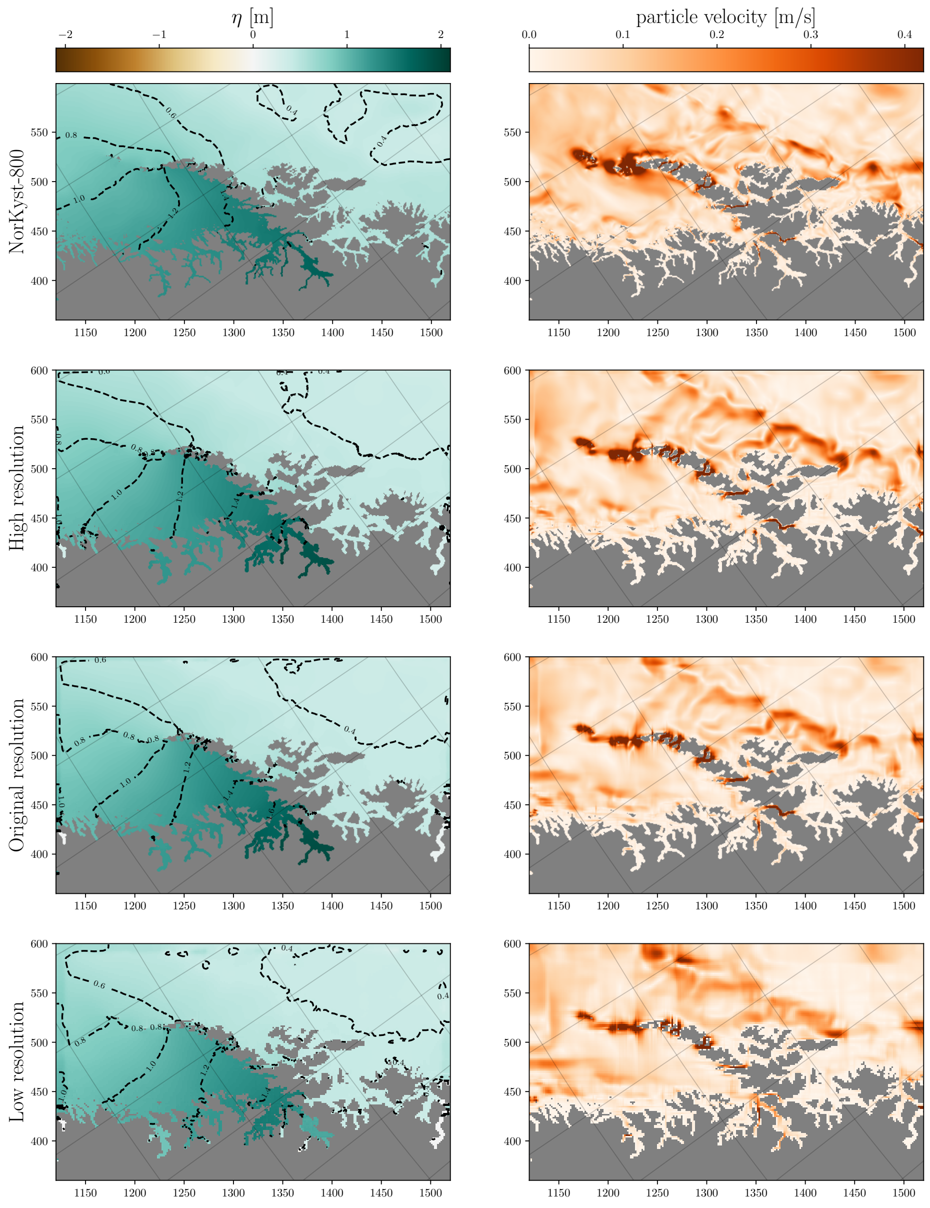


Figure 10: Case 2 around the Lofoten archipelago after 23 hours, comparing the reference solution from NorKyst-800 with our simulations using three different grid resolutions. The values on the x- and y-axes are in km relative to the location in the complete domain. Our sea-surface levels (left column) are similar but slightly higher within Vestfjorden compared to the reference solution. From the particle velocities (right), we see that we get stronger currents than the reference. The level of details in our simulations are stronger with higher grid resolution, whereas the low-resolution simulation blurs the structures and are influenced by grid effects.

solution, but we can instead more clearly see a consistent coastal current with natural irregularities. This is consistent with what we can expect using a barotropic model. These results also show a larger area with high particle velocity around the smallest islands in our simulations, possibly due to the reconstructed bathymetry, which widens some straits by the landmask erosion.

5.3 Case 3: Norway

Our third case consists of almost the complete domain covered by the NorKyst-800 model. We leave out the 25 outermost cells and use these as boundary conditions. With the complete domain, we cover a large enough area to see the Kelvin waves forming in the middle of the domain and travelling north.

Figure 11 shows the simulation results after 23 hours in the same manner as the earlier figures. We now see from the sea-surface level (left column) that there is a low tide in the middle of the domain, whereas there are high tides both in the north at the Russian border and in the Oslo fjord in the lower left corner. The contour lines are similar for all three grid-resolutions, and the particle velocities (right column) are in general consistent with the reference solution from NorKyst-800. Due to the scale of the domain, it is harder to compare the fine details in Figure 11, but the large scale features from NorKyst-800 are found in both the high and original grid resolutions. As we have seen in the other cases, the low-resolution simulation maintains less details. It should be noted that by zooming into the same areas simulated in Case 1 and 2, we have seen that the results from running the complete domain are similar within each resolution as to the results shown in Figures 9 and 10.

Table 3 reports the run time for simulating this case on a laptop, a desktop, and a server GPU. The laptop contains a dedicated GeForce 840M GPU, which is powerful for a laptop but on the low end of the GPU performance spectrum. It completes the simulation in slightly less than 54 minutes for the original grid resolution of NorKyst-800, which is comparable to the time required for 256 CPUs to run the NorKyst-800 simulation. On a desktop with a slightly old GeForce 780GTX gaming GPU, the same simulation is carried out in approximately 10 minutes. Tesla P100 is a GPU often found in supercomputers, thus belonging to the very high-end of the performance (and price) spectrum, and runs the 23-hour forecast in approximately 3 minutes. Note that for the high-resolution scheme, the time step is on average just 0.46 seconds, meaning that the results shown in Figures 9–11 are obtained after 180 000 time steps, using only single-precision floating-point operations.

5.4 Tidal forecast verification and validation

The oceanographic phenomena best captured and maintained by the shallow-water equations are arguably tides and storm surges. We therefore use Case 3 to generate tidal forecasts for five locations along the Norwegian coast (see Figure 8) and compare these with the hourly values from the NorKyst-800 dataset. The locations have been chosen based on the availability of actual sea-level observation stations, and we use these to show the realism in the forecasts generated by both NorKyst-800 and the current model. It should be noted that the sea level variation depends on more physical processes than those modeled by NorKyst-800 and our code, and we therefore expect a discrepancy between the forecasts and the observations. For us, it is therefore most relevant to use the NorKyst-800 result as a reference solution. The five locations (see Figure 8) are Bergen and Honningsvåg close to the open sea; Oslo and Narvik located

Table 3: Timing results different resolutions of running the complete Norwegian coast (Case 3) with the original, high and low resolution on different computer types.

Resolution	Num cells	Average Δt	Laptop GeForce 840M Maxwell (2014)	Desktop GeForce GTX780 Kepler (2013)	Server Tesla P100 Pascal (2016)
Low	541 875	1.85 s	9m 22s	1m 29s	47s
Original	2 167 500	0.92 s	53m 49s	10m 42s	3m 13s
High	8 670 000	0.46 s	6h 10m 09s	1h 23m 22s	23m 21s

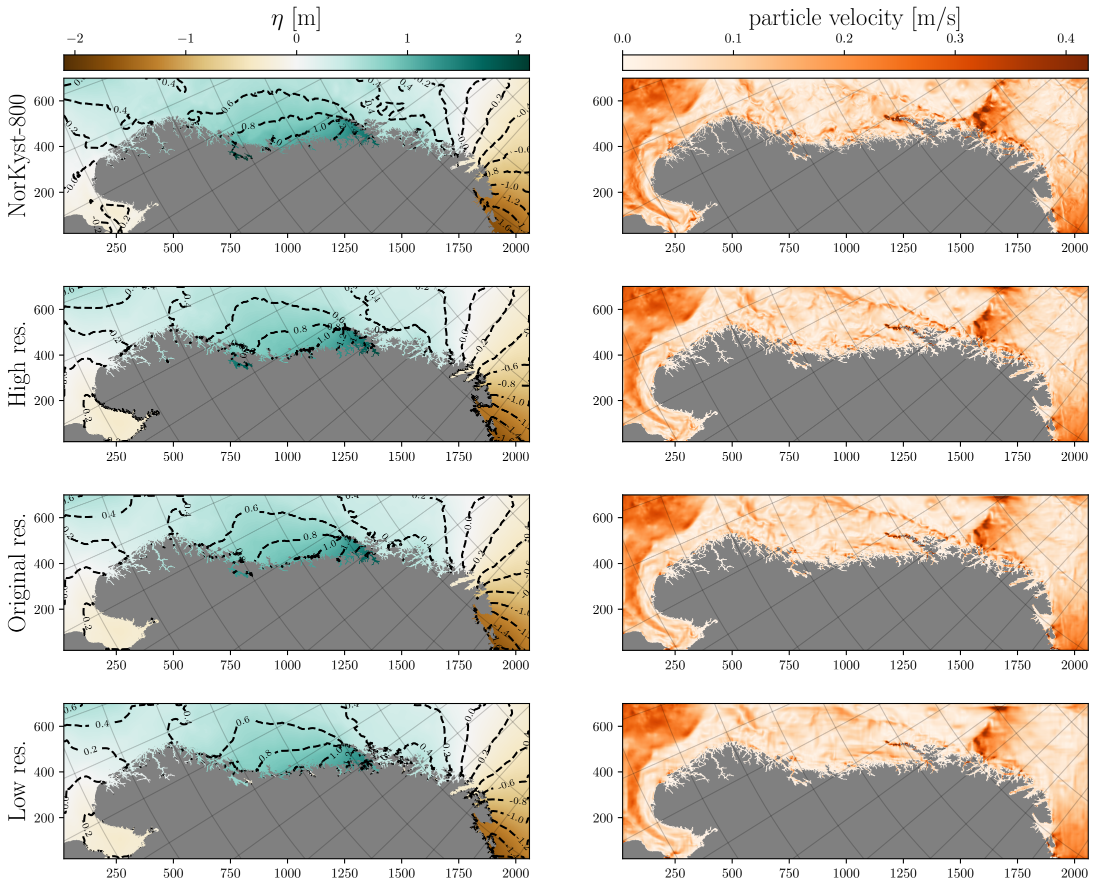


Figure 11: Simulation result for Case 3 containing almost the complete domain used by NorKyst-800 after 23 hours. The top row shows the reference solution from NorKyst-800, followed by our simulation results using three different grid resolutions below. The values on the x- and y-axes are in km relative to the location in the complete domain. The sea-surface levels (left) show how the tide varies along the coast, with low tide in the Oslofjord (lower left corner), high tide in the middle of the domain, and low tide again towards the border with Russia (lower right corner). Our results are consistent with the reference solution for all grid resolutions. The particle velocities (right column) are qualitatively similar for the reference solution (NorKyst-800), and our simulations with high and original resolution. Even at this scale, it is apparent that the low resolution grid smears the features in the ocean currents.

in the innermost parts of long fjords; and Bodø, which is affected by how the Lofoten archipelago catches the tidal waves.

Figure 12 shows the tidal forecasts generated from Case 3 for each of the five locations, sorted from south to north. First of all, we see that our forecasts at Honningsvåg are very well in accordance with NorKyst-800. The same is the case in Bergen, even though our simulations here consistently give slightly too low values for η. At Bodø, we see that NorKyst-800 gives a delay in the transition from high to low tide between 14 and 16 hours, and we can see weak

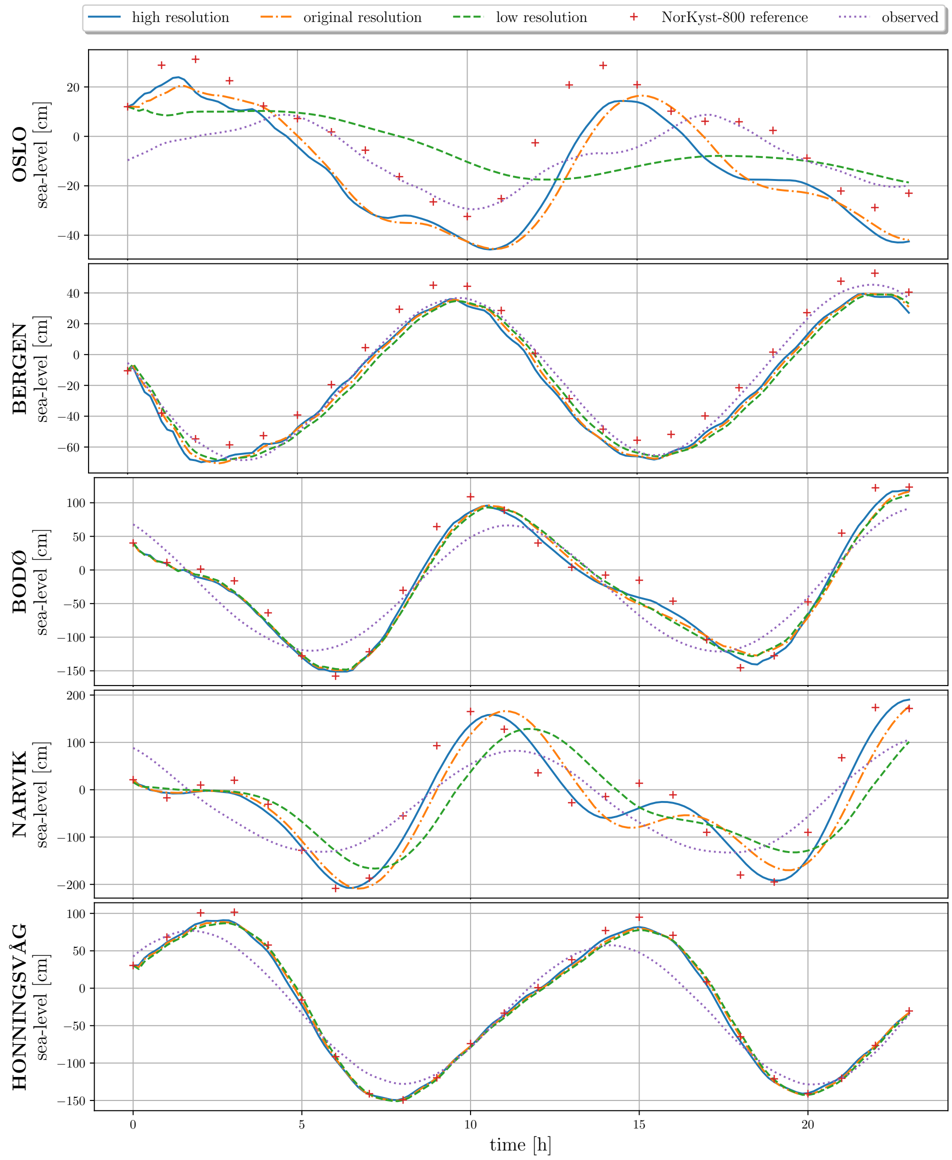


Figure 12: Tidal predictions, reference NorKyst-800 results, and official gauge measurements at selected locations generated by the simulations discussed in Case 3. All grid resolutions capture the tide as predicted by NorKyst-800 with only minor discrepancies at coastal locations, such as Bergen, Bodø and Honningsvåg. The grid resolution comes much more into play deep inside the fjords, such as at Oslo and Narvik. Here, the low-resolutions grid no longer manages to represent the bathymetry well enough for the tide to enter the fjord correctly (especially in the very narrow Oslo fjord), but we still get fair results with the high-resolution grid. The weakly dotted line is actual observed sea-surface level at the respective locations in the time range of the forecast. It should be noted that official tidal forecasts have additional physical terms, which is the main reason why the NorKyst-800 is off.

indications of this in our high-resolution simulation as well. All these locations are at the coast, which makes the tides here relatively easy to capture, but it should still be noted that they are all protected by islands. The tides in Oslo and Narvik are seen to be harder to capture, as these two cities are located at the innermost parts of two long fjords, and we get larger differences between the different grid resolutions. In Narvik, we get a large improvement by using the high-resolution grid over the original and low resolutions, but overall, our results are not too far from the reference solution. Oslo is a particularly hard case, as it is positioned within a very narrow and shallow fjord. This leads to larger relative discrepancies here, even though the absolute difference is roughly the same as in Narvik. We see that the shape of our high-resolution result is a bit different than for the reference solution. The tidal signal predicted by the low-resolution grid in Oslo is very weak, as the fjord almost becomes closed off at this resolution. Finally, note that our solutions are well behaved already from the start, meaning that we manage to initialize our simulations in a balanced state.

6 Summary

We have presented a GPU simulation framework suitable for real-world oceanographic applications. The framework is based on a modern high-resolution finite-volume method for the shallow-water equations [6] with new adaptations and extensions to handle moving wet-dry fronts, land mask, bottom friction, wind forcing, varying north vector, varying latitude, and external boundary conditions. The numerical algorithm is also improved to facilitate for efficient implementation on GPUs using single-precision floating-point operations and includes an efficient reformulation of a problematic recursive term. The framework is designed to be initialized from operational ocean forecasts issued by the Norwegian Meteorological Institute from a three-dimensional ocean model.

We have presented second-order grid convergence for a synthetic but challenging benchmark containing complex topography and varying north vector. We have also shown that the computational performance follows the expected weak scaling.

Finally, we have validated the simulation framework through three different real-world cases along the Norwegian coastline, initialized from ocean forecasts produced by the NorKyst-800 operational model with an $800 \text{ m} \times 800 \text{ m}$ horizontal resolution. Our results demonstrate that the framework manages to maintain fine rotational structures, especially when increasing the grid resolution to $400 \text{ m} \times 400 \text{ m}$. On a coarse grid (1600 m grid cells), however, the features are lost and the simulations get dominated by grid effects. The tidal forecasts show that we are able to predict the observed tides relatively well at coastal locations on any of the three grid resolutions used, when compared to the NorKyst-800 reference data. We see, however, that within long fjords, our results are improved when increasing the grid resolution, compared to using the original horizontal resolution of the NorKyst-800 model.

Acknowledgements

This research has been funded by the Research Council of Norway under grant number 250935 (GPU Ocean). The GPU Ocean core project team consists of Göran Broström, Kai Christensen, Knut-Andreas Lie, and Martin Lilleeng Sætra, and the authors are sincerely grateful for their collaboration and discussions, which have influenced and enabled this work. The authors also thank Øyvind Sætra and Jon Albretsen for feedback on drafts of the manuscript. The GPU Ocean project has received support in form of compute time on UNINETT Sigma2 - the National Infrastructure for High Performance Computing and Data Storage in Norway under project number nn9550k. The authors declare that we have no competing interests related to this work.

CRedIT author statement

A. R. Brodtkorb: Conceptualization, Methodology, Software, Investigation, Writing - Original Draft, Writing - Review & Editing, Visualization, Project administration, Supervision, H. H. Holm: Conceptualization, Methodology, Software, Investigation, Writing - Original Draft, Writing - Review & Editing, Visualization.

References

[1] A. Shchepetkin and J. McWilliams. The regional oceanic modeling system (ROMS): a split-explicit, free-surface, topography-following-coordinate oceanic model. *Ocean Modelling*, 9(4):347–404, 2005.

[2] G. Madec and the NEMO team. *NEMO ocean engine*. Note du Pôle de modélisation, Institut Pierre-Simon Laplace (IPSL), France, No 27, ISSN No 1288-1619, 2008.

[3] J. Albretsen, A. Sperrevik, A. Staalstrøm, A. Sandvik, and F. Vikebø. NorKyst-800 report no. 1: User manual and technical descriptions. Technical Report 2, Fisken og Havet, Institute of Marine Research, 2011.

[4] K. Christensen. Head of Division for Ocean and Ice, Norwegian Meteorological Institute. [personal email communication].

[5] L. Røed. *Atmospheres and Oceans on Computers*. Springer International Publishing, 2019.

[6] A. Chertock, M. Dudzinski, A. Kurganov, and M. Lukácová-Medvidová. Well-balanced schemes for the shallow water equations with Coriolis forces. *Numerische Mathematik*, Dec 2017.

[7] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing. Jupyter notebooks - a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90. IOS Press, 2016.

[8] S. Gottlieb, C.-W. Shu, and E. Tadmor. Strong stability-preserving high-order time discretization methods. *SIAM Review*, 43(1):89–112, 2001.

[9] B. van Leer. Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method. *Journal of Computational Physics*, 32(1):101 – 136, 1979.

[10] A. Kurganov, S. Noelle, and G. Petrova. Semidiscrete central-upwind schemes for hyperbolic conservation laws and Hamilton–Jacobi equations. *SIAM Journal on Scientific Computing*, pages 707–740, 2001.

[11] A. Kurganov and D. Levy. Central-upwind schemes for the Saint-Venant system. *Mathematical Modelling and Numerical Analysis*, 36:397–425, 2002.

[12] M. de la Asunción, J. Mantas, and M. Castro. Simulation of one-layer shallow water systems on multicore and CUDA architectures. *The Journal of Supercomputing*, 58(2):206–214, Nov 2011.

[13] A. Brodtkorb, M. Sætra, and M. Altinakar. Efficient shallow water simulations on GPUs: Implementation, visualization, verification, and validation. *Computers & Fluids*, 55(0):1–12, 2012.

[14] P. Parna, K. Meyer, and R. Falconer. GPU driven finite difference WENO scheme for real time solution of the shallow water equations. *Computers & Fluids*, 161:107 – 120, 2018.

[15] X. Qin, R. LeVeque, and M. Motley. Accelerating an adaptive mesh refinement code for depth-averaged flows using GPUs. *Journal of Advances in Modeling Earth Systems*, 11(8):2606–2628, 2019.

[16] NVIDIA. NVIDIA CUDA C programming guide version 10.1, 2019.

[17] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih. PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation. *Parallel Computing*, 38(3):157–174, 2012.

[18] H. Holm, A. Brodtkorb, and M. Sætra. GPU computing with Python: Performance, energy efficiency and usability. *Computation*, 8(1):4, 2020.

[19] A. Brodtkorb, T. Hagen, K.-A. Lie, and J. Natvig. Simulation and visualization of the Saint-Venant system using GPUs. *Computing and Visualization in Science*, 13(7):341–353, 2010.

[20] F. Váňa, P. Düben, S. Lang, T. Palmer, M. Leutbecher, D. Salmond, and G. Carver. Single precision in weather forecasting models: An evaluation with the ifs. *Monthly Weather Review*, 145(2):495–502, 2017.

[21] A. Kurganov. Finite-volume schemes for shallow-water equations. *Acta Numerica*, 27:289–351, 2018.

[22] A. Kurganov and G. Petrova. A second-order well-balanced positivity preserving central-upwind scheme for the Saint-Venant system. *Communications in Mathematical Sciences*, 5(1):133–160, 03 2007.

[23] W. Large and S. Pond. Open ocean momentum flux measurements in moderate to strong winds. *Journal of Physical Oceanography*, 11(3):324–336, 1981.

[24] A. Brodtkorb, C. Dyken, T. Hagen, J. Hjelmervik, and O. Storaasli. State-of-the-art in heterogeneous computing. *Journal of Scientific Programming*, 18(1):1–33, May 2010.

[25] J. Xie, L. Bertino, F. Counillon, K. Lisæter, and P. Sakov. Quality assessment of the TOPAZ4 reanalysis in the Arctic over the period 1991–2013. *Ocean Science*, 13(1):123–144, 2017.

[26] H. Davies. A lateral boundary formulation for multi-level prediction models. *Quarterly Journal of the Royal Meteorological Society*, 102(432):405–418, 1976.

[27] H. Holm, A. Brodtkorb, G. Broström, K. Christensen, and M. Sætra. Evaluation of selected finite-difference and finite-volume approaches to rotational shallow-water flow. *Communications in Computational Physics*, 27(4):1234–1274, 2020.

[28] M. Sætra, A. Brodtkorb, and K.-A. Lie. Efficient GPU-implementation of adaptive mesh refinement for the shallow-water equations. *Journal of Scientific Computing*, 63(1):23–48, April 2015.

Paper V

Data Assimilation for Ocean Drift Trajectories Using Massive Ensembles and GPUs

Håvard Heitlo Holm, Martin Lilleeng Sætra, André Rigland Brodtkorb

Accepted for publication in the proceedings of the conference
Finite Volumes for Complex Applications IX, 2020

Data Assimilation for Ocean Drift Trajectories Using Massive Ensembles and GPUs

Håvard H. Holm, Martin L. Sætra, and André R. Brodtkorb

Abstract In this work, we perform fully nonlinear data assimilation of ocean drift trajectories using multiple GPUs. We use an ensemble of up to 10000 members and the sequential importance resampling algorithm to assimilate observations of drift trajectories into the underlying shallow-water simulation model. Our results show an improved drift trajectory forecast using data assimilation for a complex and realistic simulation scenario, and the implementation exhibits good weak and strong scaling.

Key words: Particle filters, finite-volume methods, shallow-water simulations

MSC (2010)**:** 62M99, 60G35, 35L65, 65M08, 76B15, 68N19

1 Introduction

We present a proof-of-concept framework for performing fully nonlinear data assimilation of ocean drift trajectories into a shallow-water model. Forecasting drift trajectories in the ocean is an integral part of offshore preparedness services, and the forecasts are used in, e.g., search and rescue operations, oil spill tracking, and operations involving large floating structures [1]. Our approach is to use massive ensembles of simplified ocean models and assimilate observations using a particle filter based on the sequential importance resampling algorithm [2]. We first generate a massive ensemble of perturbed ocean states and simulate each ensemble member forward in time until we have an observation. Next, we use the particle filter to discard ensemble members that match poorly with the observation, and then reinitialize

Håvard H. Holm
SINTEF Digital, Mathematics and Cybernetics, Norway, e-mail: havard.heitlo.holm@sintef.no
Norwegian University of Science and Technology, Department of Mathematical Sciences, Norway,

Martin L. Sætra and André R. Brodtkorb
Norwegian Meteorological Institute, Norway, e-mail: {martinls, andreb}@met.no
Oslo Metropolitan University, Department of Computer Science, Norway,

the discarded members based on the simulated states that have a good match. We continue the simulation until the next available observation and repeat the process.

Particle filters as used here are embarrassingly parallel and require synchronization only when resampling individual ensemble members. We therefore use MPI to distribute ensemble members to different nodes, and each member is simulated forward in time using a modern explicit finite-volume scheme. The scheme is implemented on the GPU in a massively data-parallel fashion and includes all the complex source terms required for oceanographic simulations of real-world domains [3].

Our experiments show significantly increased forecast skill compared to both deterministic and Monte Carlo simulations, and we are able to run experiments with 10 000 ensemble members on the Nvidia DGX-2 server, which has 16 GPUs [4]. The implementation exhibits good weak and strong scaling and can be extended with more complex perturbation methods with minimal effort.

2 Data assimilation of ocean drift observations

Sequential importance resampling is an example of a particle filter for fully nonlinear data assimilation (see the recent review paper by Vetra-Carvalho et al. [5] on ensemble-based data assimilation techniques). A benefit of the algorithm is that, contrary to e.g., the ensemble Kalman filter [6], it does not manipulate variables of individual simulations that match well with available observations. This means that the resulting ensemble contains ocean states that are consistent with respect to the physics of the model. Furthermore, particle filters do not make any assumptions on linearity in the physical model or Gaussian probability distributions. However, the algorithm requires a large number of ensemble members as the probability that an individual ensemble member matches an observation is small. In fact, the required number of members increases exponentially with the number of observations [7, 2]. This means that it is most suitable for nonlinear problems with few observations, which is the typical situation for our target application area.

General ocean circulation models, such as ROMS [8], conserve mass, three dimensional momentum, salinity, and temperature, and operational setups typically require large computational resources to run even a single simulation. Herein, we investigate using simplified ocean models through the two-dimensional shallow-water equations, which were used operationally in the early days of computational oceanography [9]. These simplified ocean models are suitable for short-term forecasts in which the ocean can be modeled as a barotropic fluid, and they can be efficiently simulated using GPUs. A further challenge is that even the operational models often have limited forecast abilities due to by uncertain initial conditions, model parameters and forcing. By using a simplified model instead of the full three dimensional equations, we can afford to run a much larger ensemble of perturbed ocean states that can give us a more detailed description of the uncertainties in ocean forecasts that can be used as a complement to the current operational methods [10].

We have developed a GPU-based simulation framework that uses operational ocean forecasts for initial and boundary conditions, bathymetry, and forcing [3].

The framework is an extension of the high-resolution, central-upwind finite-volume scheme proposed by Chertock et al. [11], which is well-balanced with respect to steady states in which the Coriolis force balances a non-zero momentum and water surface displacement (the geostrophic balance). The scheme uses H as the water depth, η as the deviation from mean sea level, and hu and hv as the momentum along the abscissa and ordinate, respectively. We can perturb this ocean state using the approach in [12], in which we first generate a smooth random field, $\Delta\eta$, for each ensemble member, representing deviations of the ocean surface elevation. We continue by computing the momentum required to balance this perturbation, namely

$$\Delta hu_{j,k} = -\frac{gH_{j,k}}{f_{j,k}}\frac{\Delta\eta_{j,k+1} - \Delta\eta_{j,k-1}}{2\Delta y}, \qquad \Delta hv_{j,k} = \frac{gH_{j,k}}{f_{j,k}}\frac{\Delta\eta_{j+1,k} - \Delta\eta_{j-1,k}}{2\Delta x}, \quad (1)$$

and finally add these perturbations to the state variables.

Using the perturbations from (1), we generate an ensemble of ocean states and use the ensemble $\psi^n = \{\psi_0^n, \psi_1^n, ..., \psi_N^n\}$ at time t_n as an approximation to the probability density function (pdf) $p(\psi^n)$ of our ocean state. If we have an observation y^n of part of the state (e.g., one drift trajectory), we can improve the probabilistic forecast by using the conditional pdf $p(\psi^n|y^n)$. Using Bayes theorem, $p(\psi^n|y^n) = p(y^n|\psi^n)p(\psi^n)/p(y^n)$, we can write $p(\psi^n|y^n)$ as a weighted ensemble[1]:

$$p(\psi^n|y^n) \propto \sum_{i=1}^{N} \frac{p(y^n|\psi_i^n)}{\sum_{j=1}^{N} p(y^n|\psi_j^n)} \delta(\psi^n - \psi_i^n) = \sum_{i=1}^{N} w_i^n \delta(\psi^n - \psi_i^n), \quad (2)$$

in which δ is the Dirac's delta function. The weights, w_i^n, reflect how well ensemble member i matches the observation, and members with very low weights have a negligible contribution to $p(\psi^n|y^n)$. Sequential importance resampling therefore discards members with low weights and duplicates members with high weights to maintain a higher sample density in the high-probability areas.

A challenge with sequential importance resampling is the so-called curse of dimensionality, as we are operating in a very high-dimensional space. The particle filter is prone to ensemble collapse, in which the ensemble quickly reduces into only a very few significant states and thereby only has marginally better predictive skill than a purely deterministic simulation [7, 2]. This means that we need a much larger number of ensemble members compared to the number of observations. A major benefit, however, is that the particle filter makes no changes to the states of individual ensemble members during the data assimilation phase. This means that the perturbation strategy is not limited by the data assimilation method.

Fig. 1 gives an overview of the ensemble prediction system used in this paper. We use the sea-surface elevation and vertically integrated ocean currents from the operational ocean forecast provided by the ROMS-based NorKyst-800 model system [13] as initial and boundary conditions, and give an independent perturbation to each ensemble member to represent the uncertainty in the initial condition. Furthermore, we also use the same bathymetry and wind forcing as NorKyst-800. The

[1] We have ignored the marginal probability as it only serves as a normalization constant.

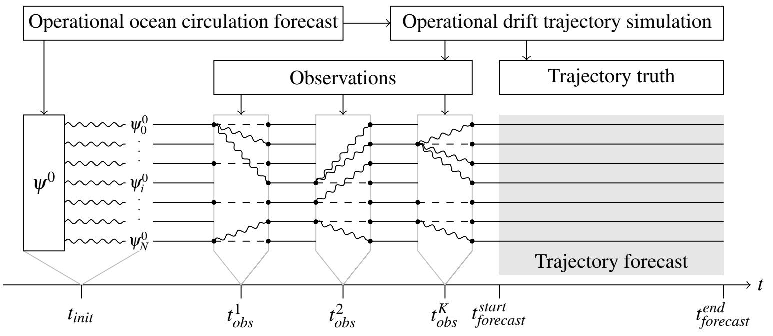


Fig. 1: Algorithmic overview of the simulation, data assimilation and drift trajectory forecast. Straight lines are deterministic simulation, wiggly lines are perturbations, and dashed lines show ensemble members that are kept during the resampling phase.

true drift trajectories are generated by OpenDrift [14] using the vertically integrated ocean currents from the hourly NorKyst-800 data, which means that the underlying physical model for the simulated truth is significantly different from our much simpler shallow-water model. From these drift trajectories, we estimate the underlying direction and velocity of the ocean currents and use this as an observation in the particle filter. We assume that the observations contain a Gaussian error with standard deviation σ and that they are independent from each other. The weight of each ensemble member is then computed as

$$w_i = \alpha \cdot \exp\left(-0.5\left(\frac{||\text{obs} - \text{sim}||}{\sigma}\right)^2\right), \tag{3}$$

in which we use the Euclidean norm to compute the distance between the observed and simulated momentum. Furthermore, α is the normalization constant such that $\sum_{i=1}^{N_e} w_i = 1$. There are several strategies for choosing which ensemble members to discard and duplicate (see [2] and references therein), and we use the residual resampling scheme [15]. Between observation times, each ensemble member runs independently and deterministically, which means that we need to perturb the duplicated ensemble states during the resampling stage.

3 Results

We test our ensemble prediction system using a domain along the coast of Northern Norway. The domain consists of 315×630 grid cells with 800 m horizontal resolution, which is the same horizontal resolution as the operational ocean circulation model that we use for initial and boundary conditions. We run 48 hours of data assimilation and use the final observed positions for the drifters as initial positions for

24 hour trajectory forecasts. To avoid that the ensemble collapses during resampling, we need to balance the number of drifters we observe with the ensemble size. Here we run experiments with 1000 and 10000 ensemble members and limit ourselves to four drifters. All experiments are run on an Nvidia DGX-2 server, equipped with 16 Tesla V100 GPUs and two CPUs, each with 24 cores.

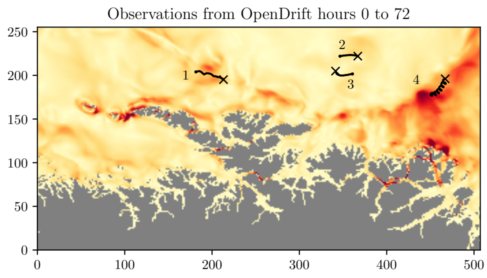


Fig. 2 Drift trajectories of four drifters over a three day period shown in the computational domain used in all our experiments. Red and yellow colors indicates strong and weak currents, respectively. Dots mark start positions and crosses end positions, and the values along the axes are in km.

Ensemble forecasts of drift trajectories We run three different experiments to illustrate the effect of data assimilation on forecasting of drift trajectories. The first is a Monte-Carlo experiment, meaning that we do not assimilate any observations during the first 48 hours. The second and third experiments are with data assimilation, and we use observations to run a particle filter every 30 minutes and 5 minutes, respectively. Fig. 2 shows the domain and the drift trajectories used as the truth.

Fig. 3 shows the forecasted drift trajectories for the four drifters in each of the three ensemble experiments with 1000 members, compared to a single deterministic forecast in green (dashed) and the truth in red (dash-dotted). The results show variable impact from data assimilation between the drifters. We see most positive effect for drifters three and four, and marginal improvement for drifter two, whereas the forecast for drifter one seems to be worse with data assimilation. The initial forecast for the first hour for drifter one, however, is significantly improved by the data assimilation, but our model is unable to capture the downward turn shortly into the forecast. This makes the ensemble perform worse in the long run when compared to the deterministic forecast. The results for drifters three and four show improvements when using observations in intervals of five minutes compared to 30 minutes. Finally, Fig. 4 shows how increasing the ensemble size to 10000 members significantly improves the forecast for drifter 2. Increasing the ensemble size increases the sampling of the pdf, and thereby also the chance that the true state is better represented by the ensemble.

Weak and strong performance scaling We evaluate the ensemble-level parallel performance by measuring the time spent in the data assimilation and forecasting parts of the code, running one hour of data assimilation with observations every five minutes and one hour of drift forecast. Note that the forecast contains no communication and should therefore show close to perfect scaling, whereas the data assimilation includes serial resampling and communication. For the weak scaling

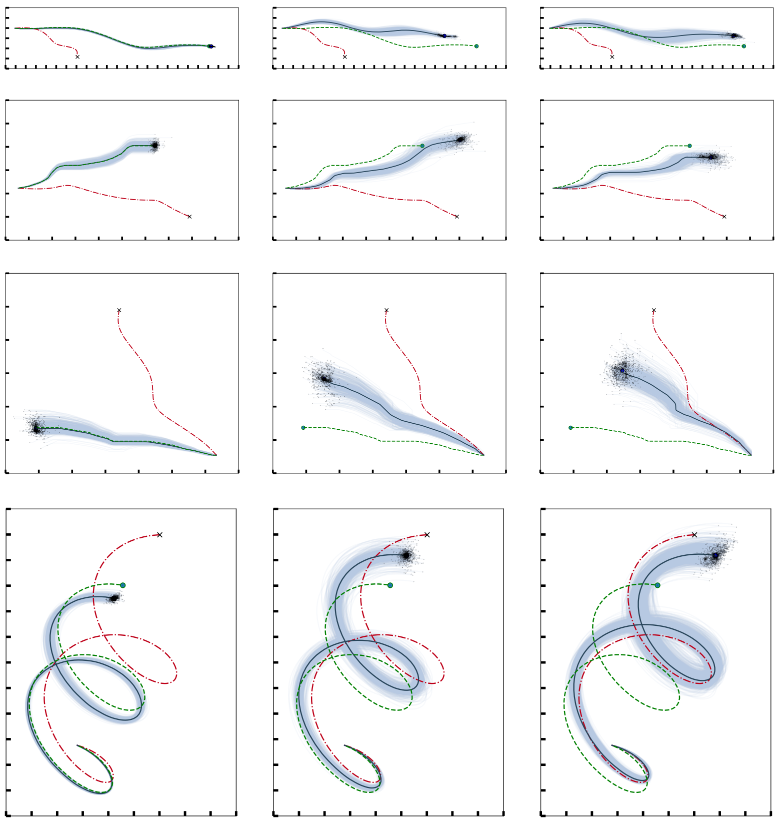


Fig. 3: Ensemble forecasts of drift trajectories with 1000 members in light blue, with the red (dash-dotted) line representing the truth, the green (dashed) line as the deterministic forecast, and the dark blue line as the ensemble mean. The four drifters are shown in separate rows, with the columns representing the three different experiments. From left to right: Monte Carlo without data assimilation, assimilation of observations every 30 minutes, and assimilation of observations every 5 minutes. The distance between the markers along the axis are one km.

experiment, we fix the per-process ensemble size at 20 members and increase the number of processes from one to 16. In the strong scaling experiment, the global ensemble size is fixed at 960 members and we vary the number of processes on which they are distributed. Each process utilizes one GPU. The results are shown in Fig. 5, with both experiments showing a 14x speedup by using 16 GPUs for the forecast. The speedup for data assimilation is nearly as good as the forecast, which means that the data assimilation does a good job preserving the parallel performance.

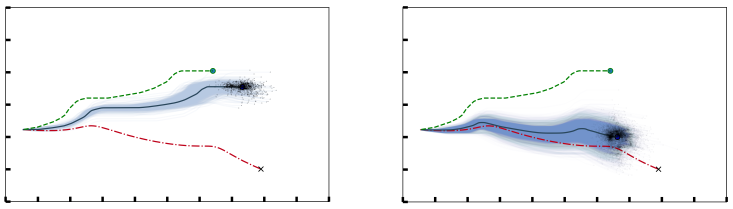


Fig. 4: Ensemble forcasting with 1000 ensemble members (left) and 10000 members (right). When using 10000 ensemble members, the forecast is significantly improved for drifter 2, while the effect is smaller for the other three drifters.

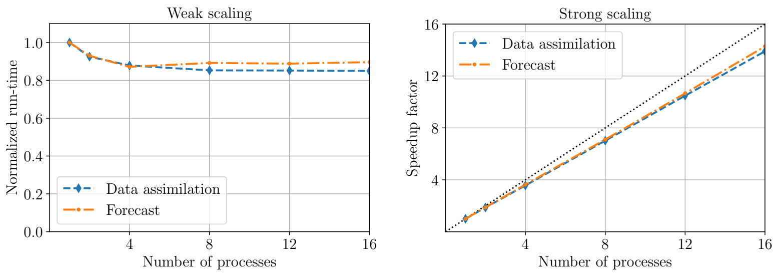


Fig. 5: The graphs show weak and strong scaling, using 1–16 GPUs on an Nvidia DGX-2 server. The top dotted line illustrates perfect strong scaling.

4 Discussion and summary

We have presented a framework for fully nonlinear data assimilation of ocean drift trajectories into a shallow-water model. The framework is implemented using the GPU for the shallow-water simulation and MPI for distribution of, and communication between, ensemble members. Our experiments show significantly increased forecast skill for simulations with data assimilation, and we are able to run over 10000 ensemble members on the Nvidia DGX-2 with 16 GPUs. The results indicate, as expected, that data assimilation with 10000 members based on 5 minute sampling of the observed drifter positions yields a better forecast than 5 minute sampling with 1000 members.

The results presented herein show that the data assimilation increases the forecast skill for three of four drifters. The forecast skill for drifter one, however, appears to be unaffected by the data assimilation, and we believe this is caused by a local predominant baroclinic ocean dynamic, which is not captured by our current simplified model. It will be an important future development to see what criteria are significant for the data assimilation to be most effective, and perhaps include a multi-layer or reduced-gravity model which can represent such dynamics better.

The simple perturbation strategy presented in this paper adds a smooth perturbation to the sea-surface level and computes the momentum required to keep the perturbation in geostrophic balance. An important extension will be to conduct more experiments with more sophisticated perturbation methods and stochastic placement of the ocean eddies, as well as perturbation of the tidal wave phase.

Acknowledgements This work is supported by the Research Council of Norway (RCN) through grant number 250935 (GPU Ocean). The computations in this paper were performed on equipment provided by the Experimental Infrastructure for Exploration of Exascale Computing (eX^3), which is financially supported by the RCN under contract 270053. The source code for the methods and experiments described in this paper is available under an GNU free and open source license released under the DOI 10.5281/zenodo.3591850.

References

1. K. Christensen, Ø. Breivik, K.-F. Dagestad, J. Röhrs, and B. Ward. Short-term predictions of oceanic drift. Oceanography, 31(3):59–67, 2018.
2. P. van Leeuwen. Particle filtering in geophysical systems. Monthly Weather Review, 137(12):4089–4114, 2009.
3. A. Brodtkorb and H. Holm. Real-world oceanographic simulations on the GPU using a two-dimensional finite volume scheme. preprint: arXiv:1912.02457, [in review], 2019.
4. NVIDIA. DGX-2/2H System user guide. Technical report, 2019.
5. S. Vetra-Carvalho, P. van Leeuwen, L. Nerger, A. Barth, M. Altaf, P. Brasseur, P. Kirchgessner, and J.-M. Beckers. State-of-the-art stochastic data assimilation methods for high-dimensional non-Gaussian problems. Tellus A: Dynamic Meteorology and Oceanography, 70(1):1–43, 2018.
6. G. Evensen. Data Assimilation: The Ensemble Kalman Filter. Springer Berlin Heidelberg, 2006.
7. C. Snyder, T. Bengtsson, P. Bickel, and J. Anderson. Obstacles to high-dimensional particle filtering. Monthly Weather Review, 136(12):4629–4640, 2008.
8. A. Shchepetkin and J. McWilliams. The regional oceanic modeling system (ROMS): A split-explicit, free-surface, topography-following-coordinate oceanic model. Ocean Modelling, 9(4):347–404, 2005.
9. E. Martinsen, B. Gjevik, and L. Røed. A numerical model for long barotropic waves and storm surges along the western coast of Norway. Journal of Physical Oceanography, 9:1126–1138, 1979.
10. L. Røed. Documentation of simple ocean models for use in ensemble predictions. Part I: Theory. Technical report, Norwegian Meteorological Institute, 2012.
11. A. Chertock, M. Dudzinski, A. Kurganov, and M. Lukácová-Medvidová. Well-balanced schemes for the shallow water equations with Coriolis forces. Numerische Mathematik, 2017.
12. H. Holm, M. Sætra, and P. van Leeuwen. Massively parallel implicit equal-weights particle filter for ocean drift trajectory forecasting. Journal of Computational Physics: X, 6:100053, 2020.
13. J. Albretsen, A. Sperrevik, A. Staalstrøm, A. Sandvik, and F. Vikebø. NorKyst-800 report no. 1: User manual and technical descriptions. Technical Report 2, Institute of Marine Research, 2011.
14. K.-F. Dagestad, J. Röhrs, Ø. Breivik, and B. Ådlandsvik. OpenDrift v1.0: a generic framework for trajectory modelling. Geoscientific Model Development, 11(4):1405–1420, 2018.
15. J. Liu and R. Chen. Sequential Monte Carlo methods for dynamic systems. Journal of the American Statistical Association, 93:1032–1044, 1998.