12-2021

A Parallelized and Layered Model for the Shallow-Water Equations

Alexander Stevens
afsteve@clemson.edu

A PARALLELIZED AND LAYERED MODEL FOR THE SHALLOW-WATER EQUATIONS

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Mathematical Sciences

---

by
Alexander F. Stevens
December 2021

Accepted by:
Dr. Qingshan Chen, Committee Chair
Dr. Sibusiso Mabuza
Dr. Fei Xue

Abstract

An energy- and enstrophy-conserving and optimally-dispersive numerical scheme for the shallow-water equations is accelerated through implementation in the GPU environment. Previous research showed the viability of the numerical scheme under standard shallow-water test cases, but was limited in applications by computation time constraints. We overcome these limitations by parallelizing the numerical computation in the GPU environment. We also extend the capabilities of the implementation to support not just a single shallow-water layer, but multiple. These improvements significantly expand the range of tests that can be used to exercise the model, and enable better understanding of the power of the numerical scheme at large scales.

Acknowledgements

Many thanks to Dr. Qingshan Chen for his help throughout, in understanding the theoretical model, troubleshooting code issues, and for generally offering support on this project.

Table of Contents

Abstract **2**

Acknowledgements **2**

1 Introduction **4**

2 Single-layer model parallelization **5**

2.1 Model and numerical scheme definition 5

2.2 Parallelization and acceleration 7

3 Multi-layer model implementation **12**

3.1 Problem Statement 12

3.2 Preliminary analysis: layer interactions 14

3.3 Implementation 18

3.4 Testing strategy 19

3.5 Numerical results 20

4 Discussion and conclusions **20**

References **22**

List of Figures

1 Parallelized model convergence results 11

2 Parallelized model runtime comparison 12

3 Vorticity evolution on a high-resolution grid 13

4 Multi-layer simulation times 21

1 Introduction

The shallow-water equations have a long history of use in the modelling of atmospheric and oceanic flows, where the vertical length scale (altitude or depth) is small relative to the horizontal length scale [1]. Much of the current interest in the shallow-water equations is in developing numerical schemes with conservative properties that match those of idealized geophysical flows; when approximated as inviscid, these flows conserve many quantities, such as mass, total energy, and potential enstrophy. Similarly of interest is developing numerical schemes that are able to accurately model the wave dispersion characteristics of real flows; these characteristics are critical to maintaining geostrophic balance in the flows, the balance between the Coriolis force and horizontal pressure gradients [2]. In some cases, numerical schemes possessing these conservative properties and wave dispersion characteristics have been able to produce more realistic dynamics than higher-order-accurate schemes, making them attractive subjects of study [3].

One such numerical scheme, developed by Chen et. al., is the focus of this paper [4]. This finite volume scheme conserves both total energy and potential enstrophy, as well as possesses optimal dispersive wave relations. It was designed for use with unstructured meshes, specifically the centroidal Voronoi mesh [5]. The numerical scheme is compatible with both a bounded domain as well as a global sphere; only the latter is considered in the remainder of this paper.

While the theoretical basis for the numerical scheme has been established, initial numerical simulations have been constrained in the maximum grid resolution and simulation length by the implementation of the model, which was designed to run almost entirely on the CPU for convenience [6]. Our method for overcoming this constraint is to parallelize the computation by moving the model into the GPU environment. This not only will speed up simulations of existing tests, but also enable previously infeasible tests to evaluate the model over long time scales or high grid resolutions.

One limitation of the shallow-water equations is that they neglect changes in fluid density in the vertical direction. A common response to this limitation is to model multiple layers stacked in the vertical direction, where each layer represents a region of constant density [1]. These layers interact by means of pressure gradients, and the result is a more realistic model of the flows. The Chen et. al. numerical scheme was developed in the single-layer context; a further area of interest from a numerical simulation standpoint is to extend the model to multiple layers and incorporate layer-to-layer interactions. We take the first steps towards this goal by proposing a multi-layer extension of

the numerical scheme, and by building on the GPU implementation to extend the model to simulate multiple layers at once.

The paper is structured as follows: Section 2.1 introduces the Chen et. al. model and numerical scheme. Section 2.2 provides an overview of the transition of the model implementation to the GPU environment, as well as test methods and results. Section 3 then discusses the multi-layer scheme and implementation with test methods and results.

2 Single-layer model parallelization

2.1 Model and numerical scheme definition

The basic steps of the formulaton of the continuous system used by Chen et. al. are repeated here, since they serve a useful basis for understanding the numerical model. We start with the vector-invariant form of the shallow-water equations

$$\begin{cases} \frac{\partial}{\partial t} h + \nabla \cdot (h\boldsymbol{u}) = 0, \\ \\ \frac{\partial}{\partial t} \boldsymbol{u} + hq\hat{\boldsymbol{k}} \times \boldsymbol{u} = -\nabla \left(g(h+b) + K \right) + \boldsymbol{F}, \end{cases} \tag{1}$$

where h is the fluid thickness, $\boldsymbol{u}$ is the fluid velocity, $\hat{\boldsymbol{k}}$ is the unit vector in the (locally) vertical direction, g is the acceleration due to gravity, b is the topography that makes up the bottom of the fluid layer, $\boldsymbol{F}$ is external forcing, and $K \equiv \boldsymbol{u} \cdot \boldsymbol{u}/2$ is the kinetic energy per unit volume of the flow. Finally, $q \equiv (f + \nabla \times \boldsymbol{u})/h$ is the potential vorticity, where f represents the Coriolis force due to the rotation of the Earth.

Taking the divergence and curl of the momentum equation $(1)_2$, we get the vorticity-divergence formulation of the shallow-water equations:

$$\begin{cases} \frac{\partial}{\partial t} h + \nabla \cdot (h\boldsymbol{u}) = 0, \\ \\ \frac{\partial}{\partial t} \zeta + \nabla \cdot (hq\boldsymbol{u}) = \nabla \times \boldsymbol{F}, \\ \\ \frac{\partial}{\partial t} \gamma - \nabla \times (hq\boldsymbol{u}) = -\Delta \left(g(h+b) + K \right) + \nabla \cdot \boldsymbol{F}. \end{cases} \tag{2}$$

Here $\zeta \equiv \nabla \times \boldsymbol{u}$ is the relative vorticity of the flow, and $\gamma \equiv \nabla \cdot \boldsymbol{u}$ is the divergence of the flow.

Note we are slightly abusing notation, as the curl operator for these two-dimensional flows should be $\text{curl}(\cdot) = \hat{\boldsymbol{k}} \cdot \big(\nabla \times (\cdot)\big)$, which is why the vorticity term is a scalar.

We can then use the Helmholtz decomposition for the mass flux term as follows:

$$h\boldsymbol{u} = \nabla^{\perp}\psi + \nabla\chi, \tag{3}$$

where $\nabla^{\perp} \equiv \hat{\boldsymbol{k}} \times \nabla$, ψ is the streamfunction for the flow, and χ is the velocity potential for the flow. Substituting this decomposition into (2), we get the final version of the continuous system:

$$\begin{cases} \frac{\partial}{\partial t} h + \Delta\chi = 0, \\ \\ \frac{\partial}{\partial t} \zeta + \nabla \cdot (q\nabla^{\perp}\psi) + \nabla \cdot (q\nabla\chi) = \nabla \times \boldsymbol{F}, \\ \\ \frac{\partial}{\partial t} \gamma - \nabla \times (q\nabla^{\perp}\psi) - \nabla \times (q\nabla\chi) = -\Delta\left(g(h+b) + K\right) + \nabla \cdot \boldsymbol{F}. \end{cases} \tag{4}$$

Chen et. al. then develops a numerical scheme for this system, for use with a centroidal Voronoi mesh; they use a Hamiltonian formulation of the system to derive the conservative properties. The details are omitted here, but the final form of the scheme on a global sphere, which conserves both total energy and potential enstrophy, is

$$\begin{cases} \frac{\partial}{\partial t} h_i &= -\left[\Delta_h \chi_h\right]_i, \\ \frac{\partial}{\partial t} \zeta_i &= -\frac{1}{2}\left(\left[\nabla_h \cdot \widetilde{(\widehat{q}_h \nabla_h^{\perp} \psi_h)}\right]_i + \left[\nabla_h \cdot (\widehat{q}_h \nabla_h^{\perp} \widetilde{\psi}_h)\right]_i\right) - \left[\nabla_h \cdot (\widehat{q}_h \nabla_h \chi_h)\right]_i \\ &\quad + \left[\nabla_h \times \boldsymbol{F}_h\right]_i, \\ \frac{\partial}{\partial t} \gamma_i &= \left[\nabla_h \times (\widehat{q}_h \nabla_h^{\perp} \psi_h)\right]_i + \frac{1}{2}\left(\left[\nabla_h \times \widetilde{(\widehat{q}_h \nabla_h \chi_h)}\right]_i + \left[\nabla_h \times (\widehat{q}_h \nabla_h \widetilde{\chi}_h)\right]_i\right) \\ &\quad - \left[\Delta_h \Phi_h\right]_i + \left[\nabla_h \cdot \boldsymbol{F}_h\right]_i. \end{cases} \tag{5}$$

Here i is the mesh cell index, the h subscript represents a discretized variable or operator, the accent $\widetilde{\ }$ indicates a cell-vertex-defined variable, the accent $\widehat{\ }$ indicates a cell-edge-defined variable, and no accent indicates a cell-center-defined variable. Note that (5) applies only when the domain is a global sphere; for a bounded domain, the divergence equation contains some extra terms for the

boundary cells. For full details, refer to [4]. Note we've introduced the notation

$$\Phi_h = g(h_h + b_h) + \widehat{h}_h^{-2}\left(\left|\nabla_h^\perp \psi_h\right|^2 + \left|\nabla_h \chi_h\right|^2 + \widehat{\nabla_h^\perp \widetilde{\psi}_h} \cdot \nabla_h \chi_h + \nabla_h^\perp \psi_h \cdot \nabla_h \widetilde{\chi}_h\right), \tag{6}$$

for the so-called geopotential.

The numerical scheme is advanced using a 4th-order Runge-Kutta timestepping scheme, and at each iteration the coupled, elliptic system

$$\begin{cases} \zeta_h = \nabla_h \times \left(\widehat{h}_h^{-1} \nabla_h^\perp \psi_h\right) + \dfrac{1}{2}\left(\nabla_h \times \widetilde{\left(\widehat{h}^{-1} \nabla_h \chi_h\right)} + \nabla_h \times \left(\widehat{h}_h^{-1} \nabla_h \widetilde{\chi}_h\right)\right), \\ \gamma_h = \dfrac{1}{2}\left(\nabla_h \cdot \left(\widehat{h}_h^{-1} \nabla_h^\perp \widetilde{\psi}_h\right) + \nabla_h \cdot \widetilde{\left(\widehat{h}_h^{-1} \nabla_h^\perp \psi_h\right)}\right) + \nabla_h \cdot \left(\widehat{h}^{-1} \nabla_h \chi_h\right), \end{cases} \tag{7}$$

must be solved for ψ_h and χ_h; this system is the numerical equivalent of the definitions for vorticity and divergence in terms of the Helmholtz decomposition (3). The system can be made symmetric by scaling each side by the cell areas. To ensure the uniqueness of the solution, we arbitrarily assign the values of ψ_h and χ_h at cell 0 to be 0. The Flexible Conjugate Gradient method with an Alegbraic Multigrid (AMG) preconditioner is used to solve this system on the GPU.

2.2 Parallelization and acceleration

2.2.1 Methodology

The numerical simulations performed by Chen et. al. were all done in a CPU environment. An obvious area of improvement to offset increases in simulation time as the result of grid refinement was to parallelize the computation by moving to a GPU environment. Chen took the first step in this direction by moving the elliptic solver, the most computationally- and time-intensive step in the scheme, into the GPU environment. This hybrid CPU/GPU approach offered some improvements in computation time over the CPU implementation, as seen in Figure 2. However the hybrid approach still left a lot of computational work to the CPU, and introduced new bottlenecks in the form of uploading and downloading large arrays to and from the GPU multiple times for each timestep.

We addressed these shortcomings by completing the transition of the numerical scheme into the GPU environment. The original implementation by Chen was done in Python, with use of the `numpy` module and its `scipy` extension to handle all of the numerical operations on the CPU (not including

the solver) [7][8]. This meant that all of the vectors and matrices used in the scheme were stored as `numpy` and `scipy` objects on the CPU; in order to move the scheme onto the GPU, all of this data had to be uploaded into GPU memory. To accomplish this, we used the `cupy` and `cupyx.scipy` Python modules [9]. These modules were designed to replicate much of the functionality of `numpy` and `scipy` in the GPU environment, so by converting all of the numerical scheme data to `cupy` and `cupyx.scipy` objects, we moved all the data (and subsequent numerical operations) onto the GPU. The uploading of data was done exclusively during the initialization of the scheme, so no further CPU to GPU data transfers were required once the scheme was initialized. The only data transfer in the opposite direction, from the GPU to CPU, while the scheme was running was periodic downloads of summary data for logging purposes. Thus the scheme was able to run completely in the GPU environment, which led to huge computation time improvements, as shown in Section 2.2.3.

One of the goals of the transition was to maintain backwards compatibility, so the numerical simulations could be carried out in either the CPU or GPU environment depending on the user's preference. This added the extra challenge of avoiding introducing redundancy to accommodate the two use cases, and developing a framework that would reuse code as much as possible for less complexity and better maintainability. In the case of interfacing with the solver, avoiding redundancy was essentially impossible, since the solvers on the CPU and GPU were different enough to require handling on a case-by-case basis. But the `cupy` and `cupyx.scipy` modules were designed to have many of the same functions, classes, and methods implemented as their CPU counterparts, so offered an elegant means of code reuse. Taking the `cupy`/`numpy` pair of modules as an example, depending on whether the user selected to run on the GPU or CPU, the appropriate module was imported under an agnostic name, `xp`; any object subsequently created using a `xp` function would then reside on the GPU or CPU, as desired, without having to query the environment again. Furthermore, after objects were created there was no need to distinguish between environments, since the classes shared many of the same methods in both modules. This conditional import strategy generally worked quite well, with a few exceptions for cases where the GPU modules were missing some methods used by the CPU version of the model; working around these cases required adding extra steps while initializing the model, but did not affect the model performance during simulation.

2.2.2 Test strategy

The main expectation of the GPU implementation was that it should deliver comparable numerical performance to the original CPU implementation. The performance was evaluated using the shallow-water standard test case (SWSTC) #2 prescribed by Williamson et. al. [10]. This test case consists of unforced, solid-body rotation in geostrophic balance, and therefore is steady-state. We consider only the case where the rotation is along the axis of Earth's rotation.

For SWSTC #2, the velocity field is given by

$$\boldsymbol{u} = \begin{pmatrix} u_0 \cos\theta \\ 0 \end{pmatrix}, \tag{8}$$

where the first and second coordinates represent longitudinal and latitudinal directions, and θ represents latitude. The Coriolis parameter is given by

$$f = 2\Omega \sin\theta, \tag{9}$$

and the thickness is given by

$$h = H_0 - \frac{1}{g}\left(a\Omega u_0 + \frac{u_0^2}{2}\right)\sin^2\theta. \tag{10}$$

Under these conditions, for ψ and χ as defined above, we have

$$\psi = -aH_0 u_0 \sin\theta + \frac{au_0}{3g}\left(a\Omega u_0 + \frac{u_0^2}{2}\right)\sin^3\theta, \tag{11}$$

$$\chi = 0, \tag{12}$$

and for vorticity and divergence

$$\zeta = \frac{2u_0}{a}\sin\theta, \tag{13}$$

$$\gamma = 0. \tag{14}$$

Note that for the constant values we use

$$a = 6.37122 \times 10^6 \text{ m}, \tag{15}$$

$$\Omega = 7.292 \times 10^{-5} \text{ s}^{-1}, \tag{16}$$

$$g = 9.80616 \text{ m s}^{-2}, \tag{17}$$

$$u_0 = \frac{2\pi a}{(12 \text{ days})}, \tag{18}$$

$$gH_0 = 2.94 \times 10^4 \text{ m}^2/\text{s}^2. \tag{19}$$

We simulated 5 days of the SWSTC #2 with a range of grid sizes, and calculated the error in the prognostic variables h, ζ, and γ at the end of the simulation relative to their intial states. We compare these results to the convergence results produced by the hybrid CPU/GPU model to ensure comparable numerical performance. We also compare the simulation run time of the two environments, to demonstrate the advantages of moving to the GPU environment.

Since the transition to the GPU environment enabled running the model with higher resolution grids, we also simulate SWSTC #5, zonal flow over an isolated mountain [10]. The setup for this test case is similar to SWSTC #2, except here we use $H_0 = 5960$ m, $u_0 = 20$ m/s, and we add a mountain centered at a longitude of $-90°$ and a latitude of $30°$ with height

$$h_s = h_{s_0}(1 - r/R), \tag{20}$$

where $h_{s_0} = 2000$ m, $R = \pi/9$, and $r^2 = \min\{R^2, (\lambda - 3\pi/2)^2 + (\theta - \pi/6)^2\}$.

This test case had been simulated by Chen over a period of 50 days with grid resolutions ranging from 480 km to 30 km [6]; we present results from a simulation of 50 days with a grid resolution of 15 km. The test case has no known analytical solution, so the results are more qualitative in nature, to demonstrate the power of the model at these high grid resolutions that were not previously feasible to simulate.

2.2.3 Numerical results

The convergence results for the SWSTC #2 are shown in Figure 1; the errors are based on the difference between the variable values after 5 days and their initial values. The L^2 and L^∞ errors for thickness and vorticity are relative, while the errors for divergence are absolute, since the initial

divergence is 0 everywhere. The L^2 divergence error is area-normalized. From Figure 1 it is clear that the GPU implementation is able to produce the same convergence characteristics as the hybrid CPU/GPU model.

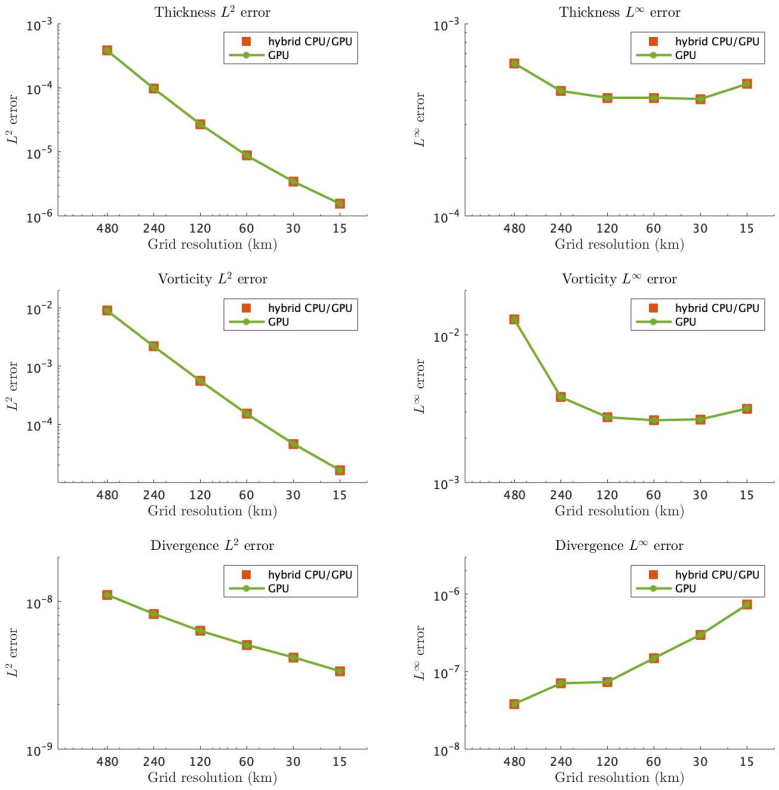
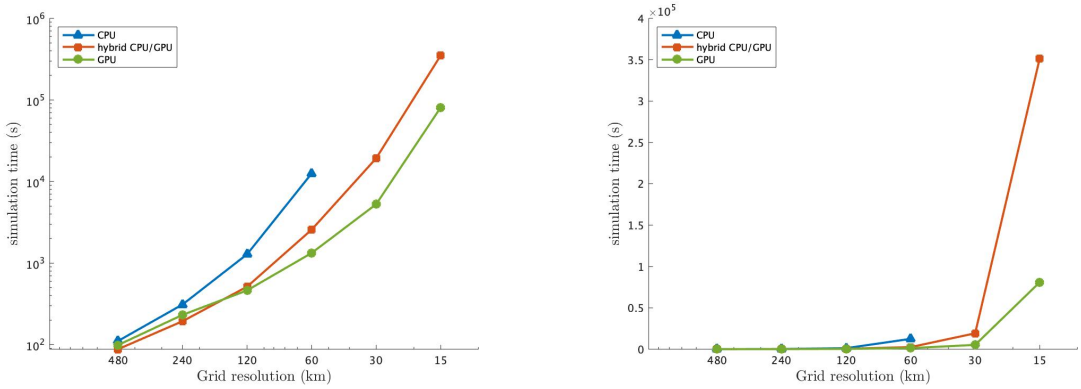


Figure 1: SWSTC #2 - 5-day convergence results for thickness, vorticity, and divergence in the L^2 and L^∞ norms, in both the hybrid CPU/GPU and the GPU environments.

The advantage of the GPU model is clear in Figure 2, which shows the wall time required to complete the 5-day SWSTC #2 simulation in the CPU, hybrid CPU/GPU, and GPU environments. For low-resolution grids, there is not much of a difference between the three approaches, but for high resolution grids the GPU environment is clearly superior. The CPU environment becomes essentially infeasibly slow above the 60 km grid. For the 15 km grid, the GPU environment offers a more than 4x computation time advantage over the hybrid CPU/GPU environment; the running times for the two cases at that grid resolution were just over 4 days and just under 1 day respectively. Note that this approximately 1 day of wall time per 5 days of simulation time ratio is likely a low estimate for



(a) log time scale

(b) linear time scale

Figure 2: SWSTC #2 - wall time required to simulate 5 days with different grid resolutions. The time axis is presented in both (2a) the log scale, to show trends, as well as (2b) the linear scale, for perspective in time differences.

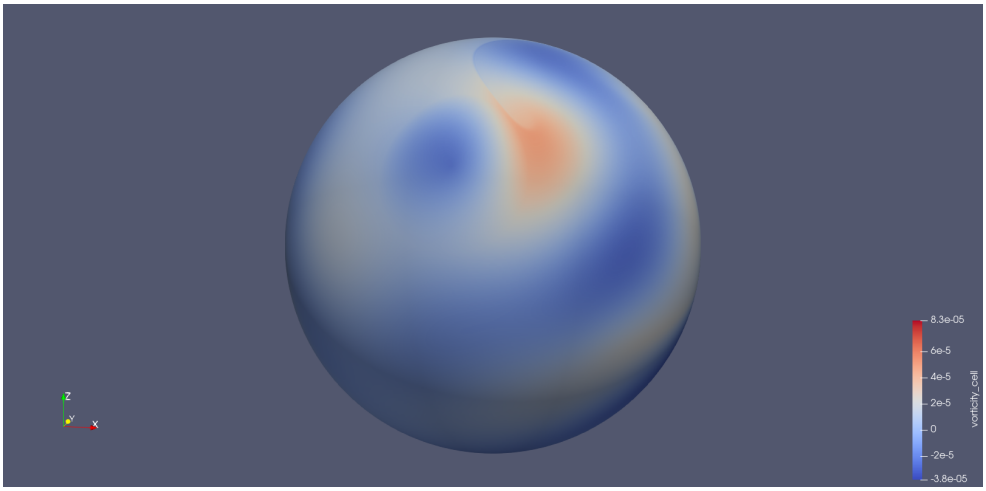
the same ratio for an arbitrary test case on the 15 km grid; since the SWSTC #2 is steady-state, and the previous values of ψ_h and χ_h are used as the initial guess in the iterative solver for the system (7) at each time step, the solver is able to produce a solution in a relatively small number of iterations. For a more dynamic test case, the number of solver iterations per time step will be larger and the model correspondingly slower.

The faster performance of the GPU implementation enabled us to run the SWSTC #5 simulation with the 15 km grid in a fraction of the time it would previously have required. Figure 3 shows a progression of plots of vorticity at the end of days 15, 20, and 25 of the simulation. As the flow evolves around the mountain topography, the model with the high resolution grid is able to capture very fine bands of high and low vorticity, as is seen after day 25. Note that the model in the simulation did not include any diffusion, so after a certain point large discontinuities (shocks) arise and model loses some of its fidelity; in our 15 km simulation, this occurs roughly around day 30.

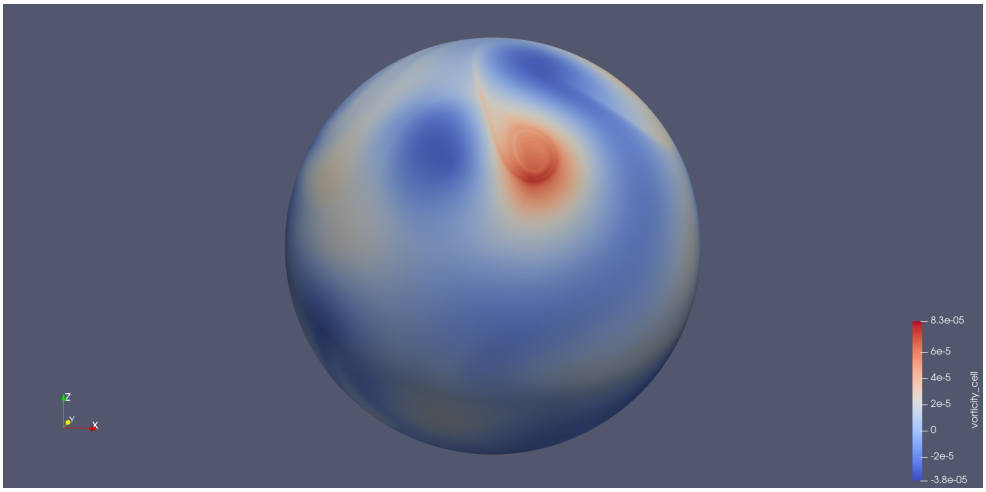
3 Multi-layer model implementation

3.1 Problem Statement

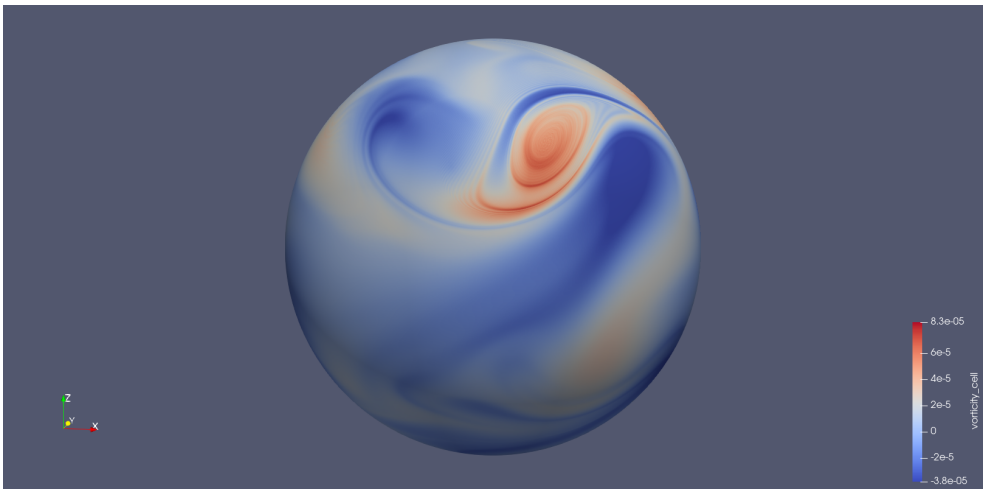
Given the success of the implementation of the model in the GPU environment, it was natural to try to extend the model further. In ocean and atmospheric modeling using the shallow-water equations, it is common practice to simulate multiple layers of shallow-water systems to represent differences



(a) Day 15



(b) Day 20



(c) Day 25

Figure 3: SWSTC #5 - evolution of vorticity using a high-resolution, 15 km grid.

in fluid density as depth or altitude changes [1]. To this end, we undertook the task of extending the model implementation to add the capability of simulating multiple layers at once.

The scope of the initial implementation work was limited to showing capability of simulating multiple layers at once, with the goal of enabling future work in simulating multi-layer systems with layer-to-layer interactions under dynamic flow conditions. As with the parallelization work, a further constraint was to make the implementation configurable to where it could be run in either the CPU or GPU environment, based on user preference. An additional expectation was that the number of layers would be configurable, not only to enable a wide range of multi-layer simulations, but also to maintain backwards compatibility with the single-layer model.

We also show preliminary analysis of layer-to-layer interactions in the continuous system, and how the numerical scheme can be extended to incorporate these interactions. Note that actually implementing these interactions was beyond the scope of the initial multi-layer implementation, but will be the subject of future study.

3.2 Preliminary analysis: layer interactions

To derive the governing equations for the multilayer system, we first start with a general form of the shallow-water equations for layer i in a multi-layer system:

$$\begin{cases} \frac{\partial}{\partial t} h_i + \nabla \cdot (h_i \boldsymbol{u}_i) = 0, \\ \frac{\partial}{\partial t} \boldsymbol{u}_i + h_i q_i \hat{\boldsymbol{k}} \times \boldsymbol{u}_i = -\nabla \left(\frac{p_i}{\rho_0} + K_i \right) + \boldsymbol{F}_i. \end{cases} \tag{21}$$

Note that this system is the same as the single-layer system (1) except for the $\nabla(p_i/\rho_0)$ term in the momentum equation. Here p_i represents the fluid pressure in layer i, and ρ_0 is the average density of all the layers; we use a Boussinesq approximation and assume that each layer's density is a small deviation from ρ_0. In the single-layer case, the fluid pressure at an arbitrary height z in the layer is given by the hydrostatic pressure equation

$$p(x, y, z, t) = g\rho_0(h(x, y, t) + b - z), \tag{22}$$

and thus the $\nabla(p/\rho_0)$ for a single layer becomes the $\nabla(g(h + b))$ term in $(1)_2$. For the multi-layer case, the hydrostatic pressure equation looks different, since we have to account for the contributions

of the all the layer thicknesses and their respective densities to calculate pressure. At an arbitrary height z in layer i, the pressure equation is

$$\begin{aligned} p_i(x,y,z,t) = g\rho_0 h_0(x,y,t) + \cdots + g\rho_{i-1}h_{i-1}(x,y,t) \\ + g\rho_i\Big(h_i(x,y,t) + \cdots h_n(x,y,t) + b(x,y) - z\Big), \end{aligned} \tag{23}$$

and consequently the $\nabla(p_i/\rho_0)$ term becomes

$$\nabla\left(\frac{p_i}{\rho_0}\right) = \nabla\left(\frac{\phi_i}{\rho_0}\right), \tag{24}$$

where

$$\phi_i = g\left[\sum_{j=1}^{i-1}\rho_j h_j + \rho_i\left(b + \sum_{j=i}^{n} h_j\right).\right] \tag{25}$$

Incorporating this into (21), we get

$$\begin{cases} \frac{\partial}{\partial t}h_i + \nabla\cdot(h_i\boldsymbol{u}_i) = 0, \\ \frac{\partial}{\partial t}\boldsymbol{u}_i + h_i q_i\hat{\boldsymbol{k}}\times\boldsymbol{u}_i = -\nabla\left(\frac{\phi_i}{\rho_0} + K_i\right) + \boldsymbol{F}_i. \end{cases} \tag{26}$$

We can show the multi-layer system (26) has similar energy- and enstrophy-conserving properties to the single-layer system when there is no external forcing. Letting Ω represent the horizontal domain, we first look at the kinetic energy budget for layer i; multiplying $(26)_1$ by $(\boldsymbol{u}_i\cdot\boldsymbol{u}_i)/2$ and $(26)_2$ by $h_i\boldsymbol{u}_i$ and summing the resulting equations we get

$$\frac{\partial}{\partial t}\left(h_i\frac{\boldsymbol{u}_i\cdot\boldsymbol{u}_i}{2}\right) + \nabla\cdot\left(h_i\boldsymbol{u}_i\frac{\boldsymbol{u}_i\cdot\boldsymbol{u}_i}{2}\right) = -\nabla\left(\frac{\phi_i}{\rho_0}\right)\cdot(h_i\boldsymbol{u}_i). \tag{27}$$

Integrating over Ω, assumed to be a global sphere (or to satisfy a no-flux boundary condition), and summing over all n layers to get the total kinetic energy of the system, we get the kinetic energy budget

$$\frac{d}{dt}\left[\sum_{i=1}^{n}\int_\Omega\left(\rho_0 h_i\frac{\boldsymbol{u}_i\cdot\boldsymbol{u}_i}{2}\right)d\boldsymbol{x}\right] = -\int_\Omega\sum_{i=1}^{n}(h_i\boldsymbol{u}_i)\cdot\nabla\phi_i d\boldsymbol{x}. \tag{28}$$

For the potential energy budget, we first introduce the notation $\eta_i = b + h_n + h_{n-1} + \cdots + h_i$ for

the vertical height of the top of layer i. The total potential energy in the system is given by

$$\int_\Omega \int_b^{\eta_1} \rho(z) g z \, dz \, d\boldsymbol{x} = \int_\Omega \frac{1}{2} g \left(\rho_1 \eta_1^2 + \sum_{i=2}^{n} (\rho_i - \rho_{i-1}) \eta_i^2 + C \right) d\boldsymbol{x}, \tag{29}$$

where C is some constant. Thus the potential energy budget is given by

$$\frac{d}{dt} \int_\Omega \frac{1}{2} g \left(\rho_1 \eta_1^2 + \sum_{i=2}^{n} (\rho_i - \rho_{i-1}) \eta_i^2 \right) d\boldsymbol{x} = \int_\Omega \frac{1}{2} g \left[\frac{\partial}{\partial t} \left(\rho_1 \eta_1^2 \right) + \sum_{i=2}^{n} \frac{\partial}{\partial t} \left((\rho_i - \rho_{i-1}) \eta_i^2 \right) \right] d\boldsymbol{x}. \tag{30}$$

Multiplying $(26)_1$ by $g\rho_1$, we can simplify the first term of the integrand in (30) to

$$\frac{\partial}{\partial t} \left(\frac{1}{2} g \rho_1 \eta_1^2 \right) = -g \rho_1 \eta_1 \sum_{j=1}^{n} \nabla \cdot (h_j \boldsymbol{u}_j), \tag{31}$$

and we can similarly simplify the other terms of the integrand by multiplying $(26)_1$ by $g(\rho_i - \rho_{i-1})$:

$$\frac{\partial}{\partial t} \left(\frac{1}{2} g (\rho_i - \rho_{i-1}) \eta_i^2 \right) = -g (\rho_i - \rho_{-1}) \eta_i \sum_{j=i}^{n} \nabla \cdot (h_i \boldsymbol{u}_i). \tag{32}$$

Plugging (31) and (32) into (30) and simplifying, we get the final potential energy budget

$$\frac{d}{dt} \int_\Omega \frac{1}{2} g \left(\rho_1 \eta_1^2 + \sum_{i=2}^{n} (\rho_i - \rho_{i-1}) \eta_i^2 \right) d\boldsymbol{x} = \int_\Omega \sum_{i=1}^{n} (h_i \boldsymbol{u}_i) \cdot \nabla \phi_i d\boldsymbol{x}. \tag{33}$$

Combining (28) and (33), we can see that the changes in kinetic and potential energy aggregated over the whole multi-layer system cancel each other out, and total energy is conserved:

$$\frac{d}{dt} \int_\Omega \left[\sum_{i=1}^{n} \left(\rho_0 h_i \frac{\boldsymbol{u}_i \cdot \boldsymbol{u}_i}{2} \right) + \frac{1}{2} g \left(\rho_1 \eta_1^2 + \sum_{i=2}^{n} (\rho_i - \rho_{i-1}) \eta_i^2 \right) \right] d\boldsymbol{x} = 0. \tag{34}$$

Taking the divergence and curl of $(26)_2$, we can put the continuous system into vorticity-divergence form:

$$\begin{cases} \frac{\partial}{\partial t} h + \nabla \cdot (h \boldsymbol{u}) = 0, \\ \frac{\partial}{\partial t} \zeta_i + \nabla \cdot (h_i q_i \boldsymbol{u}) = \nabla \times \boldsymbol{F}_i, \\ \frac{\partial}{\partial t} \gamma_i - \nabla \times (h_i q_i \boldsymbol{u}) = -\Delta \left(\frac{\phi_i}{\rho_0} + K_i \right) + \nabla \cdot \boldsymbol{F}_i. \end{cases} \tag{35}$$

From this form we can show that potential enstrophy is conserved in the multi-layer system when

there is no external forcing: rewriting $(35)_2$ in terms of potential vorticity q_i we have

$$\frac{\partial}{\partial t} q_i + \boldsymbol{u} \cdot \nabla q_i = 0. \tag{36}$$

Multiplying $(35)_1$ by $q_i^2/2$ and (36) by $q_i h_i$ and summing the result, we get

$$\frac{\partial}{\partial t}\left(h_i \frac{q_i^2}{2}\right) + \nabla \cdot \left(h_i \boldsymbol{u} \frac{q_i^2}{2}\right) = 0, \tag{37}$$

and then integrating over the domain we get the potential enstrophy budget for layer i:

$$\frac{d}{dt} \int_\Omega \frac{1}{2} h_i q_i^2 d\boldsymbol{x} = 0. \tag{38}$$

Clearly potential enstrophy is conserved within each layer, and thus is conserved for the full system.

To derive the multi-layer numerical scheme, we then use the Helmholtz decomposition for the mass flux in layer i,

$$h_i \boldsymbol{u}_i = \nabla^\perp \psi_i + \nabla \chi_i, \tag{39}$$

and write the continuous system for layer i (35)in terms of the velocity potential and streamfunction:

$$\begin{cases} \frac{\partial}{\partial t} h_i + \Delta \chi_i = 0 \\ \\ \frac{\partial}{\partial t} \gamma_i - \nabla \times \left[q_i (\nabla^\perp \psi_i + \nabla \chi_i) \right] = -\Delta \left(\frac{\phi_i}{\rho_0} + K_i \right) + \nabla \cdot \boldsymbol{F}_i \\ \\ \frac{\partial}{\partial t} \zeta_i + \nabla \cdot \left[q_i (\nabla^\perp \psi_i + \nabla \chi_i) \right] = \nabla \times \boldsymbol{F}_i \end{cases} \tag{40}$$

Since the multi-layer system (40) matches the single-layer continuous system (4) exactly except for the pressure term, we propose the following numerical scheme for cell i in layer j of the multi-layer

system:

$$\begin{cases} \frac{\partial}{\partial t} h_{i,j} &= -\left[\Delta_h \chi_{h,j}\right]_i, \\ \frac{\partial}{\partial t} \zeta_{i,j} &= -\frac{1}{2}\left(\left[\nabla_h \cdot \widetilde{(\widehat{q}_{h,j} \nabla_h^\perp \psi_{h,j})}\right]_i + \left[\nabla_h \cdot (\widehat{q}_{h,j} \nabla_h^\perp \widetilde{\psi}_{h,j})\right]_i\right) \\ &\quad - \left[\nabla_h \cdot (\widehat{q}_{h,j} \nabla_h \chi_{h,j})\right]_i + \left[\nabla_h \times \boldsymbol{F}_{h,j}\right]_i, \\ \frac{\partial}{\partial t} \gamma_{i,j} &= \left[\nabla_h \times (\widehat{q}_{h,j} \nabla_h^\perp \psi_{h,j})\right]_i + \frac{1}{2}\Bigg(\left[\nabla_h \times \widetilde{(\widehat{q}_{h,j} \nabla_h \chi_{h,j})}\right]_i \\ &\quad + \left[\nabla_h \times (\widehat{q}_{h,j} \nabla_h \widetilde{\chi}_{h,j})\right]_i\Bigg) - \left[\Delta_h \Theta_{h,j}\right]_i + \left[\nabla_h \cdot \boldsymbol{F}_{h,j}\right]_i, \end{cases} \tag{41}$$

where

$$\Theta_{h,j} = \frac{\phi_{h,j}}{\rho_0} + \frac{\left(\left|\nabla_h^\perp \psi_{h,j}\right|^2 + \left|\nabla_h \chi_{h,j}\right|^2 + \nabla_h^\perp \widehat{\widetilde{\psi}_{h,j}} \cdot \nabla_h \chi_{h,j} + \nabla_h^\perp \psi_{h,j} \cdot \nabla_h \widetilde{\chi}_{h,j}\right)}{\widehat{h}_{h,j}^2}. \tag{42}$$

Note that the only difference between this numerical scheme and the single-layer numerical scheme is we have replaced the single-layer geopotential term (6) with (42). Since we did not repeat the Hamiltonian derivation of the numerical scheme in the multi-layer context, we cannot definitively say that (41) has the same energy- and enstrophy-conserving properties as the single-layer numerical scheme (5). However we believe that the multi-layer scheme conserves both total energy and potential enstrophy aggregated across all of the layers. We leave as future work numerical evaluation of the conservative properties of the multi-layer scheme.

3.3 Implementation

In terms of implementation, an object-oriented approach of treating each layer as a separate instance of a "layer" class was considered and rejected; while elegant conceptually, this approach had the disadvantages of requiring extra memory for storing redundant data between layers, as well as requiring each layer to update its attributes separately and serially. We instead opted for the approach of converting all the structures in the model representing variable states to two dimensions, and having the ith column of the structures represent the variable states for layer i. This had the effect of essentially converting all the vector-vector element-wise operations and matrix-vector products in the single layer case to matrix-matrix element-wise operations and matrix-matrix products in the multi-layer case. These operations in the multi-layer case benefit from parallelization when running

in the GPU environment.

The conditional module import strategy employed during the parallelization of the model greatly facilitated the extension to multiple layers in both the CPU and GPU environments. An additional consideration when moving to the multi-dimensional structures for the multi-layer case is how the multi-dimensional arrays are stored; whether row-major or column-major ordering is used to store the arrays may have a significant impact on the computation time of the model. Fortunately both `numpy` and `cupy` allow the user to specify ordering, so the ordering was made configurable and each storage layout was tested as described in Section 3.4.

The solver step in the multi-layer case is handled layer-by-layer, with the variables χ_h and ψ_h for each layer depending only on that layer's thickness, vorticity, and divergence. This means that we should expect the total time needed for the solver step to increase linearly with the number of layers.

3.4 Testing strategy

We again utilize SWSTC #2 to test the multi-layer implementation. We initialize all layers with the same thickness, vorticity, and divergence as speficifed in Section 2.2.2. Given that all layers use the same model and initial state, we expect all layers to behave exactly the same; to test this, we simulated 1 day with the 480 km grid on both the CPU and GPU, with 1, 2, 4, 8, and 16 layers. As our metric of layer-to-layer variation, we use the total kinetic energy of the system after 1 day, since this tends to be a relatively volatile variable in the model, as well as the final L^2 errors in thickness, vorticity, and divergence.

To test the effect of row-major versus column-major ordering of the multi-dimensional arrays, we run the SWSTC #2 for 5 days with the 120 km grid and 16 layers on the GPU, and for 5 days with 480 km grid and 16 layers on the CPU. Since the solver step is not affected by the memory layout and we are trying to isolate the effects of the ordering, we omit the solver step for this test. Since the SWSTC #2 is steady-state, preserving the inital values of ψ_h and χ_h for all time steps is not unreasonable.

3.5 Numerical results

The results for the layer-to-layer variable comparison were as expected: the layers' total kinetic energy and L^2 errors in thickness, vorticity, and divergence after 1 day were exact matches. regardless of the number of layers. This result held for both the CPU and GPU environments, and for the 1-, 2-, 4-, 8-, and 16-layer simulations tested. In the GPU environment, while there wasn't any layer-to-layer variation in the variables examined, there was some very slight run-to-run variation; we suspect this is due to inconsistencies in how memory is allocated on the GPU at the start of simulation.

The test for the effect of of row-major versus column-major ordering did not yield conclusive results on the GPU. Each ordering was simultated on the GPU using the 120 km grid, 16 layers, and 5 days of simulation time. The simulation was repeated 5 times each, and resulted in mean run times of 133.8 and 136.2 seconds for row- and column-major ordering respectively; the difference in these sample means was not statistcally significant. For the same test on the CPU, which was run using the 480 km grid, 16 layers, and 5 days of simulation time, the row-major ordering was faster. The mean run times were 43.9 and 52.5 seconds for row- and column-major ordering respectively, which was a statistically significant difference.

With evidence that the row-major ordering is superior, we can use that configuration and examine how increasing the layer count affects simulation time using this memory layout. Figure 4 shows the time required to run 1 day of SWSTC #2 with the 120 km grid using different layer counts. As expected, the computation time increases linearly with layer count, regardless of environment. Note that the computation time advantage of the GPU environment is again obvious here.

4 Discussion and conclusions

The parallelization of the model implementation was a big success; already we have been able to simulate test cases in days that previously would have taken on the order of weeks to run. This work enables us to simulate high resolution grids that were previously infeasibly large, and that can expose properties of the model that were previously hidden by the relative coarseness of smaller grids. There are any number of test cases that can be taken as future work, for example re-running the SWSTC #5 with the 15 km grid with added diffusion to lessen the shocks.

The implementation of the model over multiple layers also enables new areas of future work. The

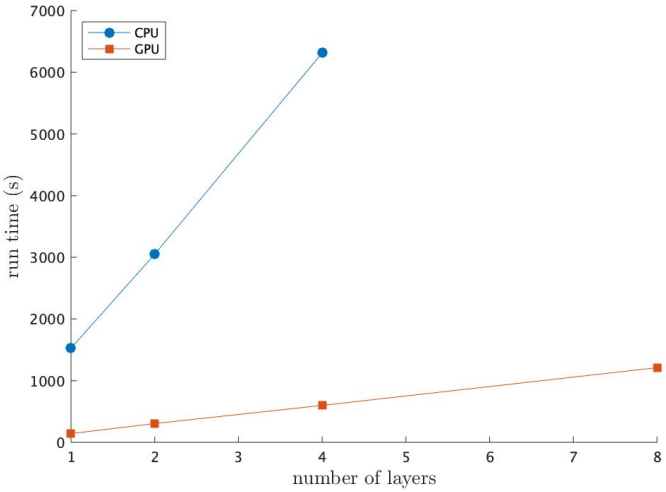


Figure 4: SWSTC #2 - wall time to simulate 1 day on the 120 km grid with different layer counts, in the CPU and GPU environments.

simplest extension of the current model is a 2-layer model using the proposed multi-layer numerical scheme with layer-to-layer interactions, which can then be extended further to a multi-layer model that more realistically captures the vertical density profile of the ocean. Sufficient accuracy of the multi-layer model would show its feasibility for use in large-scale climate simulation.

One potential constraint to new work, as the grid size and layer count grows, is a current limitation with the implementation: it can only run on a single GPU. As the memory demand of the model increases, there will need to be some special effort to optimize memory allocation or enable GPU-to-GPU communciation to enable further scaling.

References

[1] Rick Salmon. *Lectures on Geophysical Fluid Dynamics*. Oxford University Press, 1998.

[2] Akio Arakawa and Vivian R. Lamb. "Computational Design of the Basic Dynamical Processes of the UCLA General Circulation Model". In: *Methods in Computational Physics* 17.1 (1977), pp. 18–36.

[3] Akio Arakawa and Vivian R. Lamb. "A Potential Enstrophy and Energy Conserving Scheme for the Shallow Water Equations". In: *Monthly Weather Review* 109.1 (1981), pp. 18–36.

[4] Qingshan Chen, Lilu Ju, and Roger Temam. "Conservative numerical schemes with optimal dispersive wave relations: Part I. Derivation and analysis". In: *Numerische Mathematik* 149.1 (2021), pp. 43–85.

[5] Qiang Du, Vance Faber, and Max Gunzburger. "Centroidal Voronoi Tessellations: Applications and Algorithms". In: *SIAM Review* 41.4 (1999), pp. 637–676.

[6] Qingshan Chen, Lili Ju, and Roger Temam. *Conservative numerical schemes with optimal dispersive wave relations – Part II. Numerical evaluations*. 2020. arXiv: `2002.02484 [math.NA]`.

[7] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: `10.1038/s41586-020-2649-2`. URL: `https://doi.org/10.1038/s41586-020-2649-2`.

[8] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. 2001–. URL: `http://www.scipy.org/`.

[9] Preferred Networks Inc. *CuPy: NumPy & SciPy for GPU in Python v8.2.0*. Release 8.2.0. 2020. URL: `https://github.com/cupy/cupy`.

[10] David L. Williamson et al. "A standard test set for numerical approximations to the shallow water equations in spherical geometry". In: *Journal of Computational Physics* 102 (1992), pp. 211–224.