DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Benchmarking of an Ader-DG Solver for Hyperbolic Equations in the Context of Tsunami Simulations

Alexander Sytchev

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Benchmarking of an Ader-DG Solver for Hyperbolic Equations in the Context of Tsunami Simulations

Benchmarking eines Ader-DG-Lösers für hyperbolische Gleichungen am Beispiel von Tsunami-Simulationen

Author:	Alexander Sytchev
Supervisor:	Univ.-Prof. Dr. Michael Bader
Advisor:	M. Sc. Marc Marot-Lassauzaie
Submission Date:	15.11.2023

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 14.11.2023 Alexander Sytchev

Abstract

Hyperbolic partial differential equations (PDEs) govern many processes in various application fields. ADER-DG is one type of numerical solver for such PDEs. This thesis aims to assess the quality of the ADER-DG solver in the context of tsunami simulations. The so-called shallow water equations (SWE) describe such tsunami scenarios. The simulations are generated with the newly developed SWE API, which offers a straightforward configuration of shallow water scenarios. The SWE API is built on top of the ExaHyPE 2 engine for solving hyperbolic PDEs, which, in turn, is an extension of the Peano 4 framework. Peano 4 offers a parallelized, efficient traversal of the simulation space, the means to store data in that space, and the ability to specify the processing stack for the data. The experiments in this thesis show that while ADER-DG has the potential to produce high-quality results, the simulations are unstable in dry regions. Conveniently, the SWE API is built in an easily extensible way, so implementing future approaches on that topic is straightforward.

Kurzfassung

Hyperbolische partielle Differentialgleichungen (PDEs) steuern zahlreiche Prozesse in verschiedenen Anwendungsbereichen. ADER-DG ist ein numerischer Löser für solche PDEs. Ziel dieser Arbeit ist es, zu bewerten, wie sehr sich der Einsatz von ADER-DG für Tsunami-Simulationen eignet. Die sogenannten Flachwassergleichungen (im Englischen: "shallow water equations", kurz SWEs) beschreiben solche Tsunami-Szenarien. Die Simulationen werden mithilfe der neu entwickelten SWE-API generiert, die eine einfache Konfiguration von Flachwasserszenarien ermöglicht. Die SWE-API basiert auf der ExaHyPE 2-Engine zum Lösen hyperbolischer PDEs, welche wiederum eine Erweiterung des Peano 4-Frameworks ist. Peano 4 bietet eine parallelisierte, effiziente Durchquerung des Simulationsraums, und die Möglichkeit, Daten, sowie ihre Verarbeitung, zu spezifizieren. Die Experimente dieser Arbeit zeigen, dass ADER-DG zwar das Potenzial hat, qualitativ hochwertige Ergebnisse zu liefern, die Simulationen jedoch in trockenen Regionen instabil sind. Praktischerweise ist die SWE API leicht erweiterbar aufgebaut, um die Implementierung zukünftiger Ansätze zu diesem Thema zu vereinfachen.

Contents

Part I.

Introduction

Hyperbolic differential equations govern many physics, engineering, and computational science processes, ranging from fluid dynamics to electromagnetic wave propagation. Solving them efficiently and accurately is essential to understanding and predicting complex phenomena. In this context, numerical solvers have become an essential tool.

The ADER-DG method is one such solver and will be analyzed in-depth to present its design, implementation, and quality under various conditions. ADER-DG stands for **A**rbitrary high-order **DER**ivatives **D**iscontinuous **G**alerkin [**bib:ader_dg**]. It focuses on solving the differential equations in discrete cells of the simulation domain with polynomial interpolation while allowing discontinuities in the unknowns between neighboring cells. This aims to maintain the high accuracy of classical discontinuous Galerkin schemes in time and space while improving overall performance by reducing the required communication overhead of the algorithm. However, the results of the benchmarks in chapter IV show that finding a solver configuration for a stable simulation of certain scenarios can be quite challenging. The overall quality of the simulation results heavily depends on the exact initial conditions of the scenario, potentially stopping the simulation from progressing in time or breaking the calculation entirely.

One specific phenomenon explained by hyperbolic differential equations is fluid dynamics in the form of tsunamis. This presents an excellent opportunity to look at the ADER-DG solver as it solves those so-called "shallow water equations" (SWEs).

These are implemented with the help of the Peano 4 framework [**bib:peano**], together with its extension ExaHyPE 2 [**bib:exahype_swe**]. Additionally, an API explicitly tailored to the shallow water equations is built on top of ExaHyPE 2, called simply the "SWE API" [**bib:swe_api**]. It significantly simplifies the configuration pipeline of Peano 4 and ExaHyPE 2 for shallow water simulations.

The simulation of tsunamis has become increasingly relevant to minimize the damage created by these natural disasters. By combining the advancements in geology and computational science, researchers aim to develop early warning systems, enhance coastal infrastructure resilience, and find effective response strategies.

This thesis explores the ADER-DG method as it is confronted with increasingly more complex shallow water simulation scenarios. The experiments culminate in the simulation of a tsunami hitting the landmass of Japan with ADER-DG. Such a tsunami was responsible for widespread devastation of the land and civil infrastructure, as well as a meltdown of a nuclear power plant with subsequent atomic fallout, as it hit the eastern coast of Japan in 2011 [**bib:tohoku**].

First, the theory behind the ADER-DG solver and the governing differential equations is introduced in part II. Then, part III explains how the solver and the simulation scenarios are realized in software. In part IV, the quality of the simulations, as well as the performance of the software, is assessed. Lastly, a conclusion to the endeavor is given in part V.

Part II.

Theory

1. Shallow water equations

The tsunami simulations in this thesis are based on the so-called shallow water equations (SWEs) [**bib:exahype_swe**]. SWEs are differential equations defining the rules for the temporal development of the state of a fluid at any point in space, given the assumption that the horizontal width of the fluid is much larger than its depth.

Since the simulations will inevitably run on hardware with limited computing power and memory capacity, the problem has to be spatially and temporally discretized. The spatial discretization is done by splitting the domain into cells and solving the SWEs individually for all of them, considering the interaction between adjacent cells. Temporal discretization is achieved by iteratively computing finite time steps $\mathcal{T}$ of size Δt.

For the sake of simplicity, and until the introduction of the ADER-DG solver in section 2.3, each cell represents a volume of the simulation space within which all simulated variables are assumed to be constant. This is precisely the working principle of the finite volumes method, which will also be explained in section 2.2. Every cell is represented by a vector Q, which holds the unknowns of the differential equations. Following the SWE definition provided by [**bib:exahype_swe**], the Q is defined as

$$Q = \begin{pmatrix} h \\ hv_x \\ hv_y \\ b \end{pmatrix} \tag{1.1}$$

with

- x, y: Spatial (surface) coordinates
- h: Height of the water column
- v_x, v_y: Fluid velocities in x and y directions, respectively
- b: Bathymetry, i.e., the depth of the terrain compared to some base level

To advance the simulation in time, the goal is to calculate the gradient $\frac{\partial}{\partial t}Q$ and add it to the existing $Q^{\mathcal{T}}$ (at time step $\mathcal{T}$), yielding the cells' new values $Q^{\mathcal{T}+1}$ for the next time step $\mathcal{T}+1$, corresponding to $t + \Delta t$:

$$Q^{\mathcal{T}+1} = Q^{\mathcal{T}} + \Delta t \cdot \frac{\partial}{\partial t}Q(t) \tag{1.2}$$

Afterward, the interaction between neighboring cells is modeled with a Riemann flux R, resulting in the actual values $Q^{\mathcal{T}+1}$ for the next time step.

The time-stepping process is explained in detail in section 2.1.

ExaHyPE 2, the simulation engine used in this thesis, which will be explored in detail in chapter 4.2, requires the differential equations to be of the following form:

$$\frac{\partial}{\partial t}Q + \nabla \cdot F(Q) + N(Q)\nabla Q = \vec{0} \tag{1.3}$$

Here, $\nabla \cdot F$ is the divergence operator consisting of the fluxes F_x and F_y in the x and y directions, respectively:

$$\nabla \cdot F(Q) = \frac{\partial}{\partial x}F_x(Q) + \frac{\partial}{\partial y}F_y(Q) \tag{1.4}$$

The fluxes are defined as:

$$F_x(Q) = h\begin{pmatrix} v_x \\ v_x v_x \\ v_x v_y \\ 0 \end{pmatrix} = hv_x\begin{pmatrix} 1 \\ v_x \\ v_y \\ 0 \end{pmatrix} \qquad F_y(Q) = h\begin{pmatrix} v_y \\ v_y v_x \\ v_y v_y \\ 0 \end{pmatrix} = hv_y\begin{pmatrix} 1 \\ v_x \\ v_y \\ 0 \end{pmatrix} \tag{1.5}$$

$N(Q)\nabla Q$ is the so-called non-conservative product (NCP) that models the effects that do not, as the name suggests, conserve the quantities of the components of Q. Due to its dependence on the gradients ∇Q, the NCP is $N(Q)\nabla Q = 0$ within the cells. While it is ill-defined between the cells due to the discontinuity, it can be integrated within the computation of the Riemann fluxes. A detailed explanation can be found in [**bib:ncp**]. It is defined by the following equation, in which $g \approx 9.81ms^{-2}$ is the gravitational acceleration on Earth at sea level:

$$N(Q)\nabla Q = gh\begin{pmatrix} 0 \\ \frac{\partial}{\partial x}(h+b) \\ \frac{\partial}{\partial y}(h+b) \\ 0 \end{pmatrix} = gh\begin{pmatrix} 0 \\ \frac{\partial}{\partial x}H \\ \frac{\partial}{\partial y}H \\ 0 \end{pmatrix} \tag{1.6}$$

This SWE definition with equations 1.5 and 1.6 has been adapted from [**bib:exahype_swe**].

2. Solvers

The discretized SWE problem can be simulated numerically with multiple methods. This is done by defining a solver, which iteratively computes the SWE on the simulation domain, thus progressing the simulation further in time. This solver is also responsible for modeling the interactions between neighboring cells. Two kinds of solvers are studied in this thesis: Finite volumes [**bib:fv**] and ADER-DG [**bib:ader_dg**] solvers.

2.1. Time step size

The temporal progression of the simulation is represented by the time step size Δt, which is added to the total simulated time t after each iteration.

To ensure numerical stability, the so-called CFL (Courant-Friedrichs-Lewy) condition [**bib:cfl**] must be satisfied. It gives an upper bound to the time step size Δt, based on the CFL number C, the cell width w_{cell}, and the maximum eigenvalue λ_{max}. The admissible time step size Δt is defined as:

$$\Delta t \leq C \cdot \frac{w_{cell}}{\lambda_{max}} \tag{2.1}$$

For finite volumes, $C \leq 1$ must hold. For ADER-DG, on the other hand, the CFL number depends on the order ω, denoted as C_ω. Additionally, the upper bound for Δt is scaled by the factor $\frac{1}{2\omega+1}$:

$$\Delta t \leq \frac{C_\omega}{2\omega+1} \cdot \frac{w_{cell}}{\lambda_{max}} \tag{2.2}$$

For the shallow water equations, the following definition of λ_{max} was constructed with the individual eigenvalues λ_i from equation 2.55 of [**bib:leo**]:

$$\lambda_{max} = \max_i\{|\lambda_i|\} = \max\{|v_x + \sqrt{gh}|, |v_x - \sqrt{gh}|, |v_y + \sqrt{gh}|, |v_y - \sqrt{gh}|\} \tag{2.3}$$

Generally speaking, the quality of a simulation is reflected in a consistently stable progression in time. This means that it is vital to keep the eigenvalue under control. To achieve this, however, especially with ADER-DG, one has to overcome complex challenges, as presented in section 2.4.

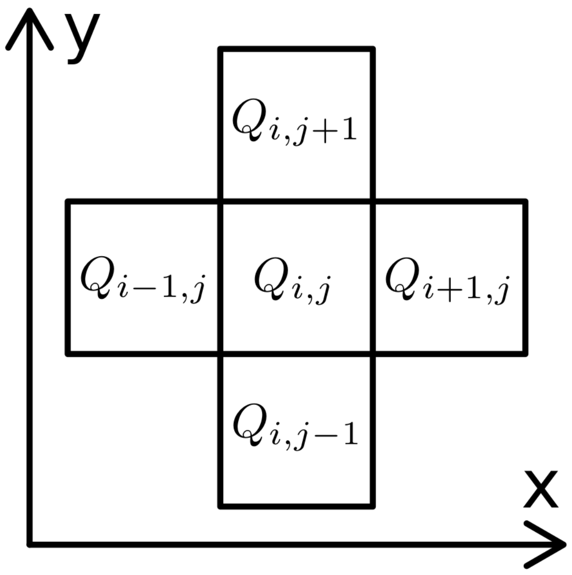


Figure 2.1.: Cell arrangement and indexing.

2.2. Finite volumes

A basic solver for hyperbolic partial differential equations is the finite volumes (FV) method. Looking at the mesh resulting from the spatial discretization, each cell represents a volume of constant value defined by the vector $Q^{\mathcal{T}}$. The superscript $\mathcal{T}$ denotes the time step corresponding to the simulation time t.

The update of the state $Q_i^{\mathcal{T}}$ of cell i to the next time step $Q_i^{\mathcal{T}+1}$, corresponding to time $t + \Delta t$, is done by solving interactions between cell i and its neighbors. These interactions are described by the flux between neighboring cells, which corresponds to a so-called Riemann problem. Let $Q_{i,j}$ be an FV cell, and $Q_{i-1,j}$ and $Q_{i+1,j}$ its left and right neighbors, respectively. Analogously, $Q_{i,j-1}$ and $Q_{i,j+1}$ are the cells to the bottom and top, respectively. Figure 2.1 demonstrates this cell arrangement.

The Riemann problem describes an initial discontinuity between the cell-wise constant Q_i and $Q_{i\pm1,j\pm1}$:

$$Q_i \neq Q_{i\pm1,j\pm1} \tag{2.4}$$

It is visualized in figure 2.2 for two arbitrary cells Q_L and Q_R.

For the sake of simplicity, the following explanation considers only the x dimension, as the concept is analogous for the y dimension. Therefore, the notation for the y dimension is omitted for the following terms. To simplify further, the non-conservative part of the SWE is omitted.

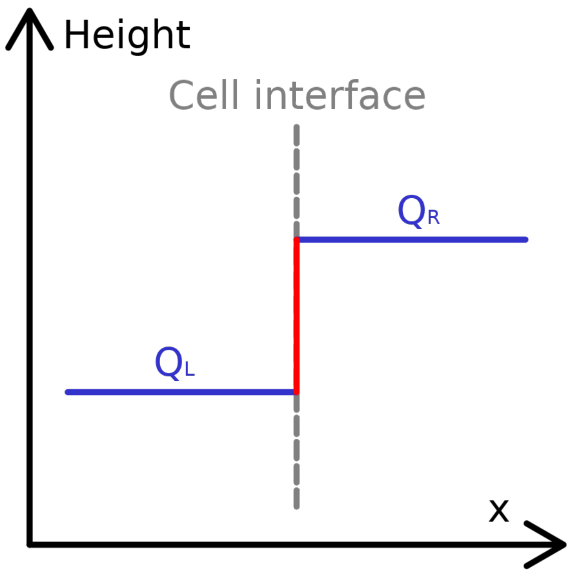


Figure 2.2.: Riemann problem at the interface between two cells due to discontinuity of Q_L and Q_R shown in red.

The solution to the Riemann problem can be approximated by modeling the numerical fluxes $\mathcal{F}(Q_i, Q_{i-1})$ and $\mathcal{F}(Q_i, Q_{i+1})$ through the left and right cell interfaces, respectively. Let the cells be squares of width w_{cell}, as set by the SWE API (see chapter 5), which generates the simulations in this thesis. The following equation then describes the cell update:

$$Q_i^{\mathcal{T}+1} = Q_i^{\mathcal{T}} - \frac{\Delta t}{w_{cell}}(\mathcal{F}(Q_i, Q_{i-1}) + \mathcal{F}(Q_i, Q_{i+1})) \tag{2.5}$$

For this thesis, the so-called Rusanov flux $\mathcal{F}_{Rus}(Q_L, Q_R)$ **[bib:rusanov_flux]** is chosen for the numerical flux. It is known to cause artificial numerical diffusion, but is rather stable and does not tend to cause oscillations **[bib:rusanov_marc]**. With Q_L and Q_R denoting the states of the cells to the left and the right of the interface, respectively. $\mathcal{F}_{Rus}(Q_L, Q_R)$ is defined as:

$$\mathcal{F}_{Rus}(Q_L, Q_R) = \frac{1}{2}(F(Q_L) + F(Q_R) - \lambda_{max}(Q_R - Q_L)) \tag{2.6}$$

2.3. ADER-DG

ADER-DG, which stands for **A**rbitrary high-order **DER**ivatives **D**iscontinuous **G**alerkin **[bib:ader_dg]**, is another type of solver for differential equations. It offers higher accuracy than conventional methods such as FV while remaining computationally efficient.

Let ω be the order of the ADER-DG solver. Instead of using one data point per cell like the finite volumes method, it discretizes the cells into a grid of $n = \bar{n}^2 = (\omega + 1)^2$ nodes, i.e., $\bar{n} = \omega + 1$ nodes per side.

With ADER-DG, the calculation of a single time step is divided into two phases: The evolution of the nodal values is first computed locally, within a single cell, in the so-called prediction phase. Afterward, these cell-local evolutions are projected to the boundary between cells and used to compute fluxes between those cells in the correction phase. The following sections explain both phases in detail.

2.3.1. Prediction phase

The prediction phase calculates the cell's local time step solution. It does this by multiplying equation 1.3 by test functions ϕ, and then by integrating it over the cell volume V in space $\bar{x}$ to obtain the so-called weak solution:

$$\int_V \frac{\partial}{\partial t} Q\phi \; d\bar{x} + \int_V \nabla \cdot F\phi \; d\bar{x} + \int_V N \nabla Q\phi \; d\bar{x} = \vec{0} \tag{2.7}$$

Additionally, this will later be integrated over time to advance the equation in time.

For simplicity, the non-conservative term $\int_V N\nabla Q\phi \; d\bar{x}$ is omitted in the following steps. A detailed explanation of the NCP can be found in [**bib:ncp**] and its incorporation into ADER-DG in [**bib:ader_dg_ncp**]. With this in mind, equation 2.8 can be simplified:

$$\int_V \frac{\partial}{\partial t} Q\phi \; d\bar{x} + \int_V \nabla \cdot F\phi \; d\bar{x} = \vec{0} \tag{2.8}$$

The product rule, when applied to the divergence term, states:

$$\nabla \cdot (F\phi) = \nabla \cdot F\phi + F\nabla \cdot \phi \Leftrightarrow \nabla \cdot F\phi = \nabla \cdot (F\phi) - F\nabla \cdot \phi \tag{2.9}$$

The so-called divergence theorem [**bib:divergence_theorem**] is used to express the volume integral of the divergence $\int_V \nabla \cdot F\phi \; d\bar{x}$ by the integral of the face over the surface S with the normal $\vec{s}$:

$$\int_V \nabla \cdot F\phi \; d\bar{x} = \oint_{\partial V} F\phi\vec{s}dS \tag{2.10}$$

Inserting this into equation 2.8 yields:

$$\int_V \frac{\partial}{\partial t} Q\phi \, d\bar{x} + \oint_{\partial V} F\phi \vec{s} dS - \int_V F\nabla \cdot \phi \, d\bar{x} = \vec{0} \tag{2.11}$$

The surface integral will be solved only in the correction phase and is therefore omitted in this phase, yielding the following purely cell-local form:

$$\int_V \frac{\partial}{\partial t} Q\phi \, d\bar{x} - \int_V F\nabla \cdot \phi \, d\bar{x} = \vec{0} \tag{2.12}$$

Rearranging the terms gives:

$$\int_V \frac{\partial}{\partial t} Q\phi \, d\bar{x} = \int_V F\nabla \cdot \phi \, d\bar{x} \tag{2.13}$$

For a polynomial representation, Lagrange polynomials are chosen using Gauss-Lobatto quadrature nodes **[bib:gauss_lobatto]**. They were chosen over the common alternative of Gauss-Legendre nodes because they yielded more stable results. These quadrature nodes allow the approximation of volume integrals using Gaussian quadrature **[bib:gauss_quad]**.

With the nodes $\bar{x}_i$ and weights w_i, the Gaussian quadrature for $\bar{x}$ in the interval $[-1, 1]$ in one dimension is:

$$\int_{-1}^{1} f(\bar{x}) \, d\bar{x} \approx \sum_{i=1}^{\bar{n}} w_i f(\bar{x}_i) \tag{2.14}$$

For an arbitrary interval $[a, b]$, $\bar{x}$ is transformed into x:

$$x = \frac{b-a}{2}\bar{x} + \frac{a+b}{2} \tag{2.15}$$

This yields the quadrature:

$$\begin{aligned} \int_a^b f(x) \, dx &= \int_{-1}^{1} f(x) \frac{dx}{d\bar{x}} \, d\bar{x} \\ &= \int_{-1}^{1} f(\frac{b-a}{2}\bar{x} + \frac{a+b}{2}) \frac{b-a}{2} \, d\bar{x} \\ &= \frac{b-a}{2} \int_{-1}^{1} f(\frac{b-a}{2}\bar{x} + \frac{a+b}{2}) \, d\bar{x} \\ &\approx \frac{b-a}{2} \sum_{i=1}^{\bar{n}} w_i f(\frac{b-a}{2}\bar{x}_i + \frac{a+b}{2}) \end{aligned} \tag{2.16}$$

Let the approximations of the volume integrals with the Gaussian quadrature be denoted as:

$$\begin{aligned} \int_V \frac{\partial}{\partial t} Q\phi \; d\bar{x} &\approx [\frac{\partial}{\partial t} Q\phi]_G \\ \int_V F\nabla \cdot \phi \; d\bar{x} &\approx [F\nabla \cdot \phi]_G \end{aligned} \tag{2.17}$$

For the complete prediction of the time step, the weak solution is integrated over one time step Δt for a given time t:

$$\int_t^{t+\Delta t} [\frac{\partial}{\partial t} Q\phi]_G \; dt = \int_t^{t+\Delta t} [F\nabla \cdot \phi]_G \; dt \tag{2.18}$$

This yields a fixed-point problem that can be solved in several ways. The PDE engine ExaHyPE2 uses so-called Picard iterations [**bib:picard**]. Again, without going into detail, let the time step solution for Q and F be denoted by:

$$\begin{aligned} \int_t^{t+\Delta t} [\frac{\partial}{\partial t} Q\phi]_G \; dt &\approx [[\frac{\partial}{\partial t} Q\phi]_G]_P \\ \int_t^{t+\Delta t} [F\nabla \cdot \phi]_G \; dt &\approx [[F\nabla \cdot \phi]_G]_P \end{aligned} \tag{2.19}$$

With the aforementioned numerical solutions to the integrals, the prediction for the local time step solution $\hat{Q}$ can be obtained.

2.3.2. Correction phase

The correction phase uses the predicted time step solution $\hat{Q}$ to model the interactions between adjacent cells. This is done by extrapolating $\hat{Q}$ onto the cell boundaries B, yielding $\hat{Q}_B$. The extrapolated values are then used to calculate the surface integral term derived in equation 2.10, which was so far left out. The surface integral is the sum of the integrals over the cell's boundaries B:

$$\oint_{\partial V} F\phi \vec{s} dS = \sum_B \int F\phi \vec{s} dS \tag{2.20}$$

Let $\hat{Q}_{B,\,L}$ and $\hat{Q}_{B,\,R}$ be the extrapolated values of arbitrary cells to the left and right of their common interface, respectively. The Rusanov flux $\mathcal{F}_{Rus}(\hat{Q}_{B,L}, \hat{Q}_{B,R})$ [**bib:rusanov_flux**] is used again for the numerical flux $\mathcal{F}$. Recall figure 2.1 for the arrangement and indexing of the cells. In this context, $\hat{Q}_{B,L}$ is the local prediction $\hat{Q}_{B,i,j}$. The neighboring cells at the boundaries B of $\hat{Q}_{i,j}$ provide their analogously extrapolated values $\hat{Q}_{B,i\pm1,j\pm1}$ as the other argument $\hat{Q}_{B,R}$ for the Rusanov flux. This results in:

$$\sum_B \int F\phi\vec{s}\, dS = \sum_{I\in\{-1,1\}} \sum_{J\in\{-1,1\}} \int \mathcal{F}_{Rus}(\hat{Q}_{i,j}, \hat{Q}_{i+I,j+J})\phi\, dS \tag{2.21}$$

This integral can again be approximated with the Gaussian quadrature:

$$\begin{aligned}&\sum_{I\in\{-1,1\}} \sum_{J\in\{-1,1\}} \int \mathcal{F}_{Rus}(\hat{Q}_{i,j}, \hat{Q}_{i+I,j+J})\phi\, dS \\ &\approx \sum_{I\in\{-1,1\}} \sum_{J\in\{-1,1\}} [\mathcal{F}_{Rus}(\hat{Q}_{i,j}, \hat{Q}_{i+I,j+J})\phi]_G\end{aligned} \tag{2.22}$$

Like the other terms of equation 2.19, this one also needs to be integrated over time with the Picard iterations:

$$\int_t^{t+\Delta t} \sum_{I\in\{-1,1\}} \sum_{J\in\{-1,1\}} [\mathcal{F}_{Rus}(\hat{Q}_{i,j}, \hat{Q}_{i+I,j+J})\phi]_G \approx [\sum_{I\in\{-1,1\}} \sum_{J\in\{-1,1\}} [\mathcal{F}_{Rus}(\hat{Q}_{i,j}, \hat{Q}_{i+I,j+J})\phi]_G]_P \tag{2.23}$$

Combining the prediction and the correction phases, the overall solution for the time step is:

$$[[\frac{\partial}{\partial t}Q\phi]_G]_P = [[F(\hat{Q})\nabla\cdot\phi]_G]_P - [\sum_{I\in\{-1,1\}} \sum_{J\in\{-1,1\}} [\mathcal{F}_{Rus}(\hat{Q}_{i,j}, \hat{Q}_{i+I,j+J})\phi]_G]_P \tag{2.24}$$

2.4. Potential problems of ADER-DG

2.4.1. Inaccurate cell boundaries

ADER-DG should produce high-order representations of the underlying solution within a cell. In the case of strong discontinuities, however, the interpolation polynomials cannot follow the desired values without inevitably producing oscillations of the water height h, as seen in figure 2.3. The figure shows a polynomial interpolation of two flat water regions of different heights. It can be seen that while the values at the nodes are accurately represented, the values outside of these data points can differ from the real solution. These inaccurate values are projected onto the cell boundaries and arrive in neighboring cells, where they are handled as a regular water movement. This significantly reduces the accuracy of the simulation. When the oscillations become

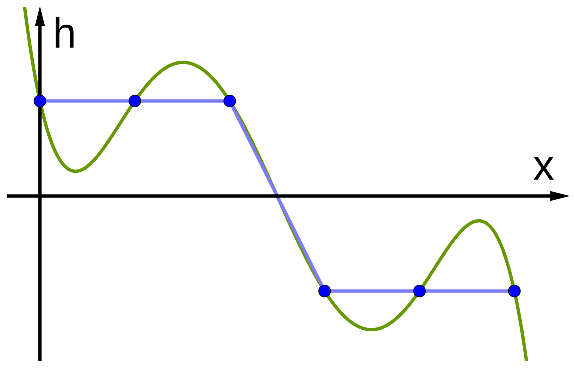


Figure 2.3.: Sharp changes in h values (blue) produce oscillations during polynomial interpolation (green).

sufficiently strong in amplitude, they cannot be numerically represented, and the algorithm produces "NaN" ("not a number") values. When NaNs are computed once, they propagate across the whole domain, breaking the simulation entirely.

2.4.2. Numerical inaccuracies

Intuitively, a body of water with a constant water level and $v_x = v_y = 0$ should stay at rest. Mathematically, the divergence of the fluxes $\nabla \cdot F$ should fully cancel out the NCP $N\nabla Q$ in this case:

$$\nabla \cdot F = -N\nabla Q \tag{2.25}$$

Using equation 1.3, this results in no change in the water state over time:

$$\frac{\partial}{\partial t} Q = \vec{0} \tag{2.26}$$

One common issue in numerical solvers, including ADER-DG, is that those terms do not entirely cancel each other out numerically due to rounding errors and the not fully accurate projection of interpolation polynomials onto the cell boundaries. This produces oscillations of the water level, which are then propagated across the domain, as explained in section 2.4.1.

2.4.3. Dry data points

Both finite volumes and ADER-DG SWE solvers have one problem in common. The numerical solution of the SWE can sometimes lead to negative values of h in shoreline regions. This, however, is mathematically impossible, assuming no groundwater and similar details, because the minimum water height is, by definition, $h = 0$ when the region is dry.

As explained in section 2.4.2, however, an initially constant water level does not mean it will stay constant throughout the numerical simulation. Since the dry regions should not contribute fluxes to their neighbors anyway, the SWE must be bypassed there to prevent them from doing so due to numerical inaccuracies. For that, the wetness threshold T_{wet} is introduced as a hyperparameter. The data point is considered dry if $h < T_{wet}$. When this happens, the flux F and the non-conservative term N are set to $\vec{0}$:

$$F = N = \vec{0}. \tag{2.27}$$

Recall that the definition of the maximum eigenvalue λ_{max} for calculation of the time step size Δt (equation 2.3). There, the velocities v_x and v_y are used. However, ExaHyPE 2 requires the calculation of λ_{max} from Q alone. The definition of Q (equation 1.1) stores the velocities as hv_x and hv_y. This leads to the following extraction step:

$$\begin{aligned} v_x &= \frac{hv_x}{h} \\ v_y &= \frac{hv_y}{h} \end{aligned} \tag{2.28}$$

Now, a problem arises with dry cells, i.e., $h = 0$:

$$\begin{aligned} v_x &= \frac{hv_x}{h} = \frac{0}{0} \\ v_y &= \frac{hv_y}{h} = \frac{0}{0} \end{aligned} \tag{2.29}$$

This division by 0 breaks the calculation of the time step size Δt and, with it, the whole simulation. To prevent this, the maximum eigenvalue is also set to 0 in dry cells, i.e., when $h < T_{wet}$:

$$\lambda_{max} = 0 \tag{2.30}$$

Together with the zeroing out of the flux and the NCP (equation 2.27), this workaround consistently stabilizes fully dry regions.

Still, it sometimes helps to apply another safety mechanism: The introduction of a lower bound of 0 to the water height h during time step postprocessing. This is a dedicated processing step for the $Q^{\mathcal{T}+1}$ vector after the time step solution for $\mathcal{T} + 1$ has been computed, yielding a new vector $Q_P^{\mathcal{T}+1}$. This new vector $Q_P^{\mathcal{T}+1}$ is then used as the input

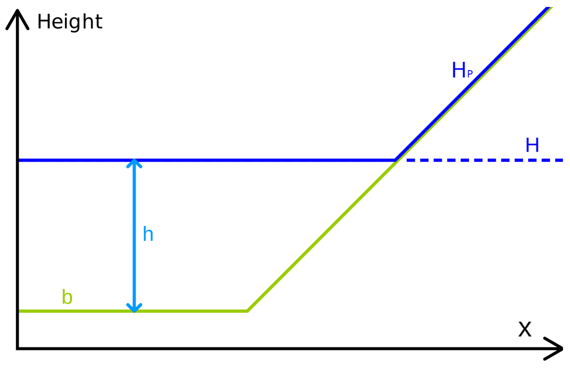


Figure 2.4.: Non-negativity postprocessing, resulting in $H_P = \max\{b,\ H\}$.

for the next iteration. With the lower bound of 0, the water height h_P, and consequently the water level H_P, are thus:

$$h_P = \max\{0, h\} \Leftrightarrow H_P = \max\{b, H\} \tag{2.31}$$

Recall the Q vector definition from equation 1.1. There, all components, including the velocities v_x and v_y, also depend on h. Taking this into account, the new vector $Q^{\mathcal{T}+1}$ must therefore be

$$Q^{\mathcal{T}+1} = \begin{pmatrix} h_P \\ h_P v_x \\ h_P v_y \\ b \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ b \end{pmatrix} \tag{2.32}$$

Figure 2.4 demonstrates the effect of the non-negativity postprocessing on the water level H with the new water level H_P in a one-dimensional context. There, H is initially constant, so the bathymetry would produce a negative h when $b \geq H$, which is fixed by the postprocessing yielding $H_P = b$ in this region.

3. Peano curve

Since the simulation domain comprises discrete cells, a method of accessing and evaluating all the cells must be implemented. Enter the space-filling curves. Space-filling curves (SFC) aim to traverse a multidimensional domain efficiently while reaching every point in that space. SFCs realize this by recursively subdividing the domain and constructing an efficient path in a predefined pattern through the resulting subdomains.

One of the main advantages of space-filling curves is their parallelizability. The available compute units (i.e., threads or CPU cores) can traverse disjoint sets of the subdomains concurrently, yielding significant speedup. This is why they are a convenient choice for the traversal of the SWE simulation domain.

A simple space-filling curve is the so-called Peano curve. Peano 4, the software framework used for this thesis, as described in section 4.1, uses this specific SFC, hence the name. For the sake of simplicity, the domain is assumed to be square. The Peano curve recursively subdivides the domain into a 3 by 3 grid of square subdomains. Each subdomain is traversed column by column, as seen in the left part of figure 3.1. The individual paths through the subdomains can be stitched together to traverse the whole domain in one efficient path. Figure 3.1 shows the resulting SFCs for recursion depths $d = 2$ and $d = 3$.

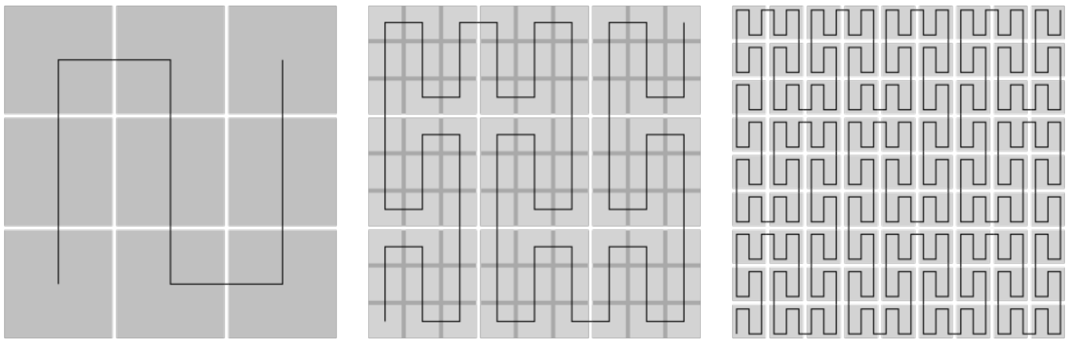


Figure 3.1.: Subdivision of the domain with a recursion depth of 1, 2, and 3 [**bib:peano_wikipedia**].

Part III.

Implementation

4. Peano 4 and the ExaHyPE 2 extension

The simulation framework used for this thesis is called Peano 4 [**bib:peano**], together with the built-in ExaHyPE 2 [**bib:exahype_swe**] extension. While Peano 4 lays the foundation for the data structures and low-level data processing, the ExaHyPE 2 extension provides the means to specify the concrete data processing needed for SWE simulations. This part of the thesis explores their workings and how they are applied to the shallow water simulation use case.

4.1. Peano 4

Peano 4 is a framework for the efficient discretization and traversal of multidimensional simulation domains. In the case of SWEs, the domains are two-dimensional. They are split up into grids of rectangular cells. To these cells, actual data and its processing methods can be attached.

4.1.1. Domain traversal

The cells in such a grid are traversed along the Peano curve, as described in chapter 3. To do that, the Peano curve is split into segments, which are mapped onto compute units, i.e., processor cores. Each core then computes the time step solution for its segment of the Peano curve and then synchronizes the boundaries with the cores responsible for the neighboring segments. The parallelizability of the Peano curve traversal enables the simulation to be run on powerful hardware, such as supercomputers, improving the runtime significantly. This is demonstrated in figure 4.1, where the different colors of the path show a potential mapping of the path segments to CPU cores. To consider the interaction between the segments, however, the data has to be synchronized across the cores, resulting in overhead.

Peano 4 supports multiple interfaces for parallelization, including OpenMP [**bib:openmp**], MPI [**bib:mpi**], as well as GPU offloading.

4.1.2. Low-level configuration

The simulation can be set up on a low level by creating a Peano 4 project in Python. It takes all hyperparameters needed for the complete definition of the simulation domain

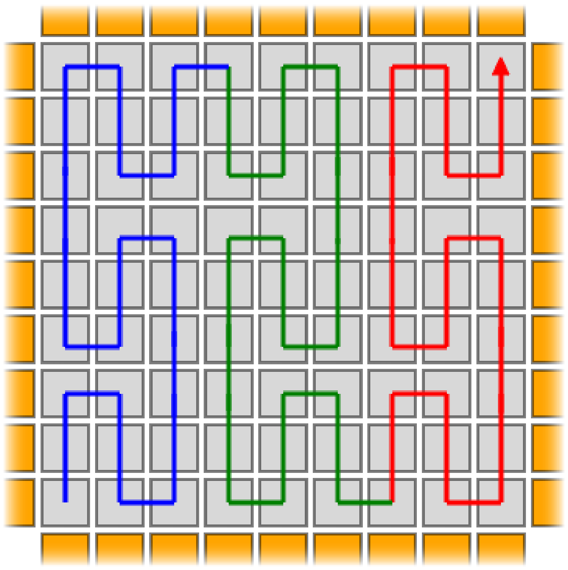


Figure 4.1.: Traversal of the 9 by 9 cells simulation domain (gray) following the Peano curve with recursion depth $d = 2$. The colors along the path show a potential segmentation for individual CPU cores. ExaHyPE 2's boundary halo is shown in orange.

and its discretization and subsequent traversal, apart from the actual processing of the cells. These hyperparameters include:

- Load balancing type
- Domain size
- Cell size
- User-defined constants

The user has to fill those stubs with code corresponding to the specific differential equation.

4.2. ExaHyPE 2

To realize the simulation of shallow water equations, an engine for solving hyperbolic PDEs is used on top of the low-level Peano 4 configuration. This engine is called ExaHype 2 [**bib:exahype_swe**] and is an extension of the Peano 4 framework. It presents an API to define the specific processing of the cell data across its internal nodes and the faces toward neighboring cells.

4.2.1. Solvers

One of the main components of the ExaHyPE 2 setup is the solver. The user can choose between numerous solver types, including, but not limited to, finite volumes and ADER-DG. The solvers can be configured extensively. Apart from the necessary hyper-parameters, such as the cell width w_{cell}, or the ADER-DG order, optional specifications can be added, for example, time step postprocessing.

The setup is performed by creating an ExaHyPE 2 project, which extends the Peano 4 project from section 4.1.2. After building the project, the user is presented with generated C++ callback function stubs. They represent the differential equation problem and follow the form of equation 1.3. These functions are:

- Initial condition
- Boundary conditions
- Flux
- Non-conservative product

Additionally, there is a function for the maximum eigenvalue of the problem, which is used to control the time step size of the simulation when the solver type implements adaptive time stepping.

4.2.2. Boundary conditions

During the time step computation, the cells' interaction with their neighbors has to be considered. However, if a cell is on the boundary, there is no adjacent cell to interact with in that direction. ExaHyPE 2 solves this problem with a halo of virtual cell faces around the domain, which is then used for the flux calculation of the boundary cells within the domain. Intuitively, one can view those virtual faces as projections of imaginary cells around the domain, shown in figure 4.1 in orange. The values of the virtual faces ($Q_{outside}$) can be calculated from the values of the boundary cells themselves (Q_{inside}). A more detailed explanation of the boundary conditions in the context of SWEs will be explained in section 5.5.

5. SWE API

Since this thesis revolves around one single differential equation, namely the SWE from chapter 1, it makes sense to create another level of convenience on top of ExaHyPE 2, which builds the complete SWE simulation after providing all necessary hyperparameters.

Usually, the user has to manually use a plethora of Peano 4 and ExaHyPE 2 API calls to configure a simulation fully. However, the different SWE scenarios explored in this thesis, or the future (see part V), differ in only the hyperparameters and the initial conditions. This paves the way for significant simplification of the configuration pipeline and is realized by an API called the "SWE API" [**bib:swe_api**].

5.1. Python configuration code

The goal of the Python configuration code is to provide an API with which the user can specify the complete set of hyperparameters for the simulation. These are used to construct a `SweBuilder` object.

An `SweBuilder` object has a single method `build()`, which handles all data passing and building of the Peano 4 project, as well as setting up all C++ code that is not scenario-specific.

5.2. Solver definition

As already mentioned, ExaHyPE 2 offers multiple solver types, such as FV and ADER-DG. From the user's perspective, each has its own interface, decreasing its interchangeability with other types. The SWE API minimizes the differences between the different solver types. Additionally, it removes the need to understand the technical intricacies of the ExaHyPE 2 solvers.

The user now only needs to input the scenario-specific parameters to a `Solver` object provided by the SWE API. This `Solver` object is then used by the `SweBuilder` to instantiate an appropriately configured ExaHyPE 2 solver object.

The SWE API also features different postprocessing algorithms, which can be given to the `Solver` objects, potentially improving the stability of the simulation. One example is the water height non-negativity postprocessing, as explained in section 2.4.3. Multiple postprocessing algorithms can be specified at once in the desired order of application. The postprocessing can be applied to the whole domain or selectively to the boundary or corner cells. The `Postprocessing` interface, which the different algorithms implement, can also be easily extended, allowing for more sophisticated postprocessing approaches in the future.

5.3. SWE implementation

As previously stated in section 4.1, the user has to manually fill in the C++ code stubs for each simulation scenario. Since the SWE code stays consistent across different scenarios, this can be worked around by filling the stubs once and saving the resulting file as a template for future builds. In subsequent builds of the ExaHyPE 2 project, this template file replaces the C++ file in the build directory that would typically contain the code stubs. When the build completes, the resulting simulation is configured to solve the SWEs. Since every solver type generates a different stub code, each one needs one such template file.

Another thing to consider is that the signature of the code stubs can vary slightly from one solver type to another. This is solved by having solver-specific template files that all implement the same SWEs and replacing the stub-containing file with the template for the desired solver. To mitigate the resulting problem that each template has to implement the SWEs from scratch, the whole SWE logic is extracted into its centralized implementation. The templates then call this single source of truth during the actual calculation. This is good practice in programming and makes maintaining and updating the SWE code significantly more accessible.

5.4. Scenario definition

For a complete description of the initial conditions of the SWE scenario, the bathymetry and the initial water height need to be specified for the whole domain. This is realized by the `SweScenario` interface. By implementing this interface for a specific scenario, the user can provide definitions of the required initial conditions.

5.4.1. Domain and cell sizes

Since the SWE API sets both the domain and the cells to be square, defining them by their widths makes sense. Recall that the traversal of the cells follows the Peano curve (see section 3, which recursively subdivides the (sub-)domain into a 3 by 3 grid. At this point, the SWE API does not support adaptive mesh refinement (AMR) Therefore, the whole domain is a grid of 3^d by 3^d square cells for a recursion depth d. To express the desired cell width w_{cell} in a compact way, the d is used together with the domain width hyperparameter w_{domain}:

$$w_{cell} = \frac{w_{domain}}{3^d} \tag{5.1}$$

5.4.2. Bathymetry definition

One prominent feature of the SWE API is the ability to import arbitrary bathymetry for the simulation in the form of NetCDF files with the help of NetCDF C++ library [**bib:netcdf**]. This allows for simulations of real-world scenarios, such as the tsunami that hit the coast of Japan in 2011, which was one of the causes of the Fukushima nuclear reactor meltdown [**bib:tohoku**], which made news across the globe. The simulation, which uses this real-world data, is explored in section 6.4.

Often, it is needed to define the bathymetry programmatically, depending on the position (x, y) in the simulation domain. An example use case is the "sloped beach" scenario, as studied in section 6.3. There, the bathymetry follows the equation $b = -0.1x$. This, too, is implemented in the SWE API, with the coordinates x and y being provided as the parameters for the bathymetry calculation.

Both methods of bathymetry definition are interchangeable in the SWE API. However, when using the NetCDF bathymetry definition, the interface `SweScenarioNetcdf` needs to be implemented instead of `SweScenario`. This is there to relieve the user of the need to implement the bathymetry manually since the NetCDF file already defines it.

5.4.3. Initial conditions

Recall the definition of the vector of unknowns $Q = (h, hv_x, hv_y, b)^T$ (see equation 1.1).

The specification of the initial water height h is done programmatically. Similar to the bathymetry definition, the coordinates x and y are provided. Additionally, the bathymetry b at the point (x, y) is given. Amongst other potential use cases, this allows for a more straightforward definition of the initial water height in terms of the water level H_0. The initial water height h_0 can now be calculated from the water level:

$$h_0 = H_0 - b \tag{5.2}$$

Having covered two of the four components of the vector Q, only the water velocities (times h), hv_x, and hv_y remain. They are set to zero, which represents an initially static water volume that the shallow water equations take over at $t = 0$:

$$hv_x = hv_y = 0 \tag{5.3}$$

5.5. Boundary conditions

As briefly stated in section 4.1, the boundary conditions are realized by a halo of virtual cell faces around the simulation domain, which is then used for the flux computation. The Peano framework provides a callback function stub with the value of the cell inside the domain Q_{inside} next to the boundary and a pointer in which to store the value of the corresponding outside cell face $Q_{outside}$.

The type of boundary conditions chosen for the SWE API are the so-called "reflecting" boundary conditions. Intuitively, this means that the domain with all its water is a closed system and that waves hitting the boundaries are reflected into the domain. This is implemented by setting the water height $h_{outside}$, as well as the bathymetry $b_{outside}$ of the virtual cell face to the values of the neighboring cell inside the domain (h_{inside}, b_{inside}) and inverting the velocities ($hv_{x,inside}$, $hv_{y,inside}$) in the corresponding direction:

$$Q_{outside} = \begin{pmatrix} h_{outside} \\ hv_{x,outside} \\ hv_{y,outside} \\ b_{outside} \end{pmatrix} = \begin{pmatrix} h_{inside} \\ -hv_{x,inside} \\ -hv_{y,inside} \\ b_{inside} \end{pmatrix} \tag{5.4}$$

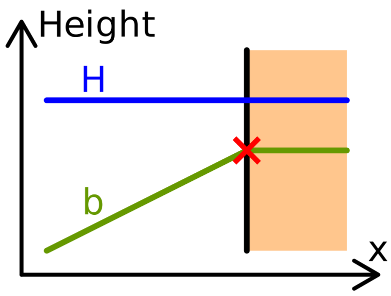
This might lead to a problem, however. When the bathymetry is a slope near the boundary, and the bathymetry of the outside cell is set to the same value as the inside cell, there is a sharp change in the gradient between the cells. Figure 5.1a highlights this gradient change with a right red cross in a cross-section of the domain. The green line represents the bathymetry b, and the blue line the water level H. The orange region is the emulated cell outside the simulation boundary. For numerical PDE solvers, such situations on the boundary might lead to instability.

To mitigate this problem, the SWE API can extend the simulation domain by a halo of flat bathymetry of width w_{halo}, as demonstrated by figure 5.1b. This is equivalent to shifting the problem from the boundary into the actual simulation domain, where the computation is more stable.

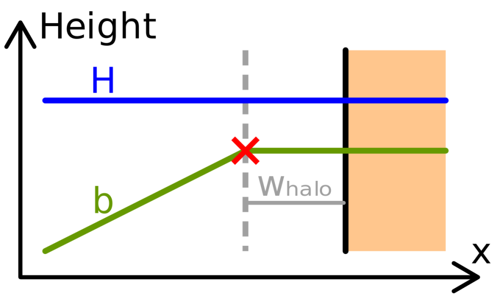
The new domain width w'_{domain} is thus

$$w'_{domain} = w_{domain} + 2 \cdot w_{halo} \tag{5.5}$$

The main downside of this approach is the enlargement of the domain size. Recall that the resolution of the simulation is anti-proportional to the cell width (see equation 5.1).



(a) A sharp change in the bathymetry gradient at the domain boundary might lead to instability.



(b) The domain is extended by w_{halo} to shift the sharp change in the bathymetry into the simulation domain. The dashed gray line is where the domain boundary originally was. This may increase the stability.

Figure 5.1.: Cross-section of the bathymetry and water level at the domain boundary. The red crosses highlight sharp changes in the bathymetry gradient. The black vertical line represents the boundary, and the orange region is Peano's emulated boundary cell.

Since the cell width w_{cell}, is defined via the subdivision depth of the domain, an increase of the domain size also increases the cell width to w'_{cell}:

$$\begin{aligned} w'_{cell} &= \frac{w'_{domain}}{3^d} \\ &= \frac{w_{domain} + 2 \cdot w_{halo}}{3^d} \\ &= \frac{w_{domain}}{3^d} + \frac{2 \cdot w_{halo}}{3^d} \\ &= w_{cell} + \frac{2 \cdot w_{halo}}{3^d} \end{aligned} \tag{5.6}$$

This, in turn, decreases the resolution. Since the halos are generally kept small compared to the total domain size, this decrease in resolution is negligible in most cases.

This flat bathymetry halo, which the SWE API provides, must not be confused with ExaHyPE 2's realization of the general boundary conditions (see section 4.2.2) using a halo of virtual cell faces around the simulation domain.

5.6. Complete definition of a simulation

Building all the previous components together, a complete simulation definition would look like this:

- Solver
  - Solver-specific properties
  - Postprocessing
- Domain width w_{domain}
- Subdivision depth d
- Bathymetry b
- Initial water height h_0
- Wetness threshold T_{wet}
- Halo width w_{halo}

The SWE API still needs some auxiliary technical parameters, but they are irrelevant to the overall simulation outcome and are thus left out in this thesis.

Figure 5.2 summarizes a simulation's configuration data flow by the SWE API and its underlying frameworks ExaHyPE 2 and Peano 4. The orange elements represent Python code, and the blue ones represent C++ code.

5.7. Additional features

5.7.1. Analysis tools

Independently of the simulation configuration, the SWE API ships with Python tools to analyze the performance of the simulations. The so-called trackers hook into the simulation output and capture the time step data t and Δt per iteration. The timestamps τ of the individual iterations can also be tracked. This data is then saved into CSV (comma-separated values) files. The included CSV parsers and plotters can then visualize the recorded data. The figures and tables in chapter 7 were generated using these analysis tools.

5.7.2. Bash scripts

Several shell commands are included to control the SWE API from the command line easily. They are a convenient way to build, run, and render (for the visualization with ParaView [**bib:paraview**]) the simulations:

- `swebuild`: Executes the SWE API building pipeline to create an executable for the simulation
- `swesim`: Runs the simulation

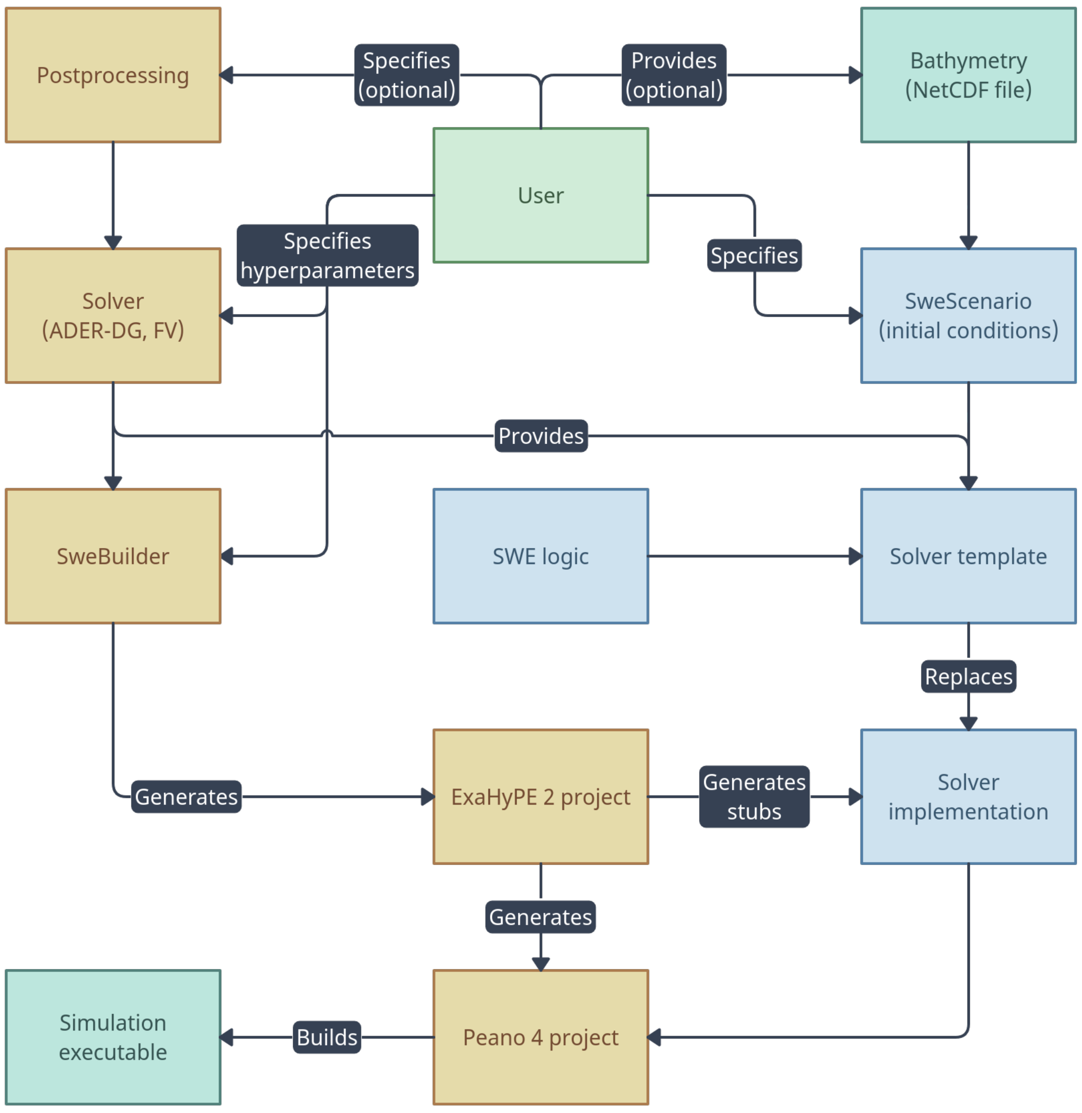


Figure 5.2.: Configuration data flow of the SWE API and its underlying frameworks Peano 4 and ExaHyPE 2, which yields an executable for the simulation. The orange elements represent Python code, and the blue ones represent C++ code.

- `swerender`: Renders the results of the simulation for the visualization with Paraview
- `swefull`: Subsequently executes the commands `swebuild`, `swesim`, and `swerender`
- `sweperf`: Runs the simulation with `swesim` and tracks the iteration data (see section 5.7.1)

For these commands to function correctly, multiple system setup steps have to be performed:

1. The commands must be made available to the environment variable `PATH`
2. The environment variable `SWE_API_PATH` must be set to the directory containing the SWE API
3. The environment variable `PEANO_PATH` must be set to the directory containing the Peano 4 implementation
4. The environment variable `PYTHONPATH` must be extended by `$SWE_API_PATH/python` and `$PEANO_PATH/python`

Part IV.

Benchmarking

The following factors are used to assess the quality of the simulations computed with an ADER-DG solver:

1. Stability of the computation
2. Shape of the resulting volumes of water
3. Performance

To evaluate the stability of the simulations, it is most interesting to look at the parts of the solver which induce the most instability. Recalling the previous sections, several limitations of ADER-DG were mentioned. This part explores multiple shallow water scenarios built to showcase these limitations and evaluates how significant they are for general applications. For comparison, some of the following scenarios are simulated with both the FV and the ADER-DG solvers provided by the SWE API.

All visualizations of the simulations were created with the software ParaView [**bib:paraview**].

6. Scenarios

It is essential to mention that all the following scenarios use reflecting boundary conditions, as stated in 5.4. Recalling section 5.4.3, the initial water velocities are always set to $v_{x,0} = v_{y,0} = 0$.

6.1. Resting lake

One of the most trivial scenarios imaginable is a volume of water without any water movement. In the context of this thesis, this is called a "resting lake."

The main goal of this scenario is to verify whether the flux and NCP terms cancel each other out, as explained in section 2.4.2. Mathematically, they should do this when the overall water level H is constant, and all velocities are $v_x = v_y = 0$.

Multiple stages of the resting lake scenario are defined to investigate this.

6.1.1. Fully dry

The most trivial scenario is the one with no water, i.e., $h_0 = 0$. Together with a bathymetry $b = 0$, the initial configuration Q_0 is therefore:

$$Q_0 = \begin{pmatrix} h_0 \\ v_{x,0} \\ v_{y,0} \\ b \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \vec{0}$$

The domain width w_{domain}, the cell width w_{cell} (defined by subdivision depth d), and the wetness threshold T_{wet} are chosen as the following:

$$\begin{aligned} w_{domain} &= 1m \\ d = 4 \Leftrightarrow w_{cell} &= \frac{w_{domain}}{3^d} = \frac{1m}{3^4} = \frac{1}{81}m \\ T_{wet} &= 0.01m \end{aligned} \tag{6.1}$$

The complete simulation configuration, therefore, looks like this:

- ADER-DG solver
  - Order $\omega = 3$
- Domain width $w_{domain} = 1m$
- Subdivision depth $d = 4$
- Bathymetry $b = 0$
- Initial water height $h_0 = 0$
- Wetness threshold $T_{wet} = 0.01m$

Note that this data representation matches the structure of the inputs for the SWE API, which allows for future reconstruction of the same scenario.

The result is, unsurprisingly, that the fluid dynamics simulation immediately reaches a time step size of $\Delta t = 0$ due to the absence of fluid.

6.1.2. Fully wet

The next step is to fully flood the terrain with still water. From now on, it makes more sense to refer to the terrain as bathymetry. The full scenario is defined as follows:

- ADER-DG solver
  - Order $\omega = 3$
- Domain width $w_{domain} = 1m$
- Subdivision depth $d = 4$
- Bathymetry $b = sin(5x) \cdot sin(5y) \cdot 1m$
- Initial water height $h_0 = 3m - b$
- Wetness threshold $T_{wet} = 0.01m$

A lake at rest should, without external forces, stay at rest. This is expressed mathematically by the mutual cancellation of the flux and NCP terms (see equation 2.25).

Figure 6.1 examines those oscillations. They are of such a low magnitude compared to the overall water height that they can be safely neglected. The SWE API can thus successfully simulate a lake at rest with ADER-DG.

6.1.3. Partially dry

The last stage of the resting lake scenario is the presence of dry cells. This scenario uses the same bathymetry as the previous one but reduces the water level such that parts of the bathymetry lie above water. The full scenario is defined as:

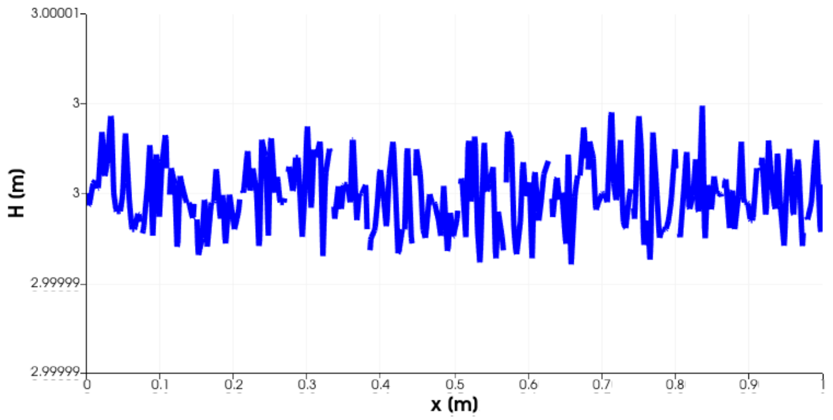


Figure 6.1.: Unnatural water level oscillations in a fully flooded resting lake scenario. Scenario parameters: ADER-DG order $\omega = 3$, $d = 4$.

- ADER-DG solver
  - Order $\omega = 3$
- Domain width $w_{domain} = 1m$
- Subdivision depth $d = 4$
- Bathymetry $b = sin(5x) \cdot sin(5y) \cdot 1m$
- Initial water height $h_0 = \max\{0,\ 0.5m - b\}$
- Wetness threshold $T_{wet} = 0.01m$

The simulation runs stably until $t \approx 0.06s$, corresponding to about 1200 iterations. Then, a problem arises, destabilizing the simulation and leading to inconsistent and, more importantly, dropping time step sizes Δt. Oscillations of the water height (and thus water level) occur in partially dry cells. These oscillations increase in amplitude over time and propagate to nearby wet cells. Eventually, they become so large that the solver is unable to represent them, turning them into NaNs ("not a number"). The NaNs are propagated in only a few iterations across the whole simulation domain, after which the time step size reaches $\Delta t = 0$, breaking the simulation entirely at $t \approx 0.1s$ after about 3300 iterations. Figure 6.2 shows those water level oscillations at $t \approx 0.09s$. The red and white area in the middle of the figure represents one of the islands, i.e., $h = 0$.

A closer look reveals that they are the strongest in those cells, where two neighboring cells are mostly dry, and the other are mostly wet. It is important to note that partially dry cells always produce stronger oscillations than fully wet ones. Usually, oscillations from one neighboring dry cell are not sufficient to cause the solver to fail. However, when the oscillations from two or more sides arrive in one cell at once, they add up to be strong enough to induce instability in that cell. From then on, the oscillations

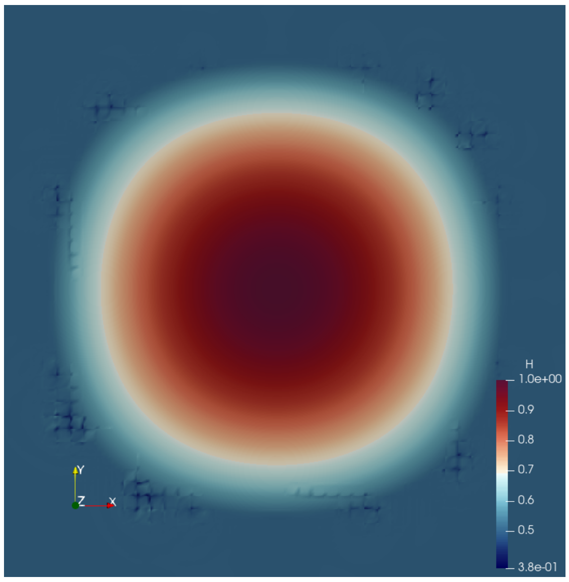


Figure 6.2.: Oscillations at partially dry cells in the resting lake scenario at $t \approx 0.09s$. The red and white area in the middle of the figure represents one of the islands, i.e., $h = 0$. Scenario parameters: ADER-DG order $\omega = 3$, $d = 4$.

increase indefinitely inside that cell until they break the simulation. Note that the cell where the oscillations from the neighbors meet must already contain water ($h > 0m$) for this phenomenon to occur.

The conclusion for this scenario is, unfortunately, that ADER-DG cannot handle water-containing cells when arranged in such a way that at least two neighbors are mostly dry.

6.2. Radial outflow

The following two scenarios are designed to isolate the effects of moving water volumes from the already observed problems of ADER-DG in resting lake scenarios. For that, radially symmetrical volumes of water are placed onto a domain at rest with a fixed water height α and flat bathymetry $b = 0$.

6.2.1. Smooth water volume

The first scenario has no sharp changes of the initial unknowns Q_0, especially water height h_0. A vertically flipped sigmoid function is used to represent the smooth slope of the water volume:

$$\sigma_\gamma(r) = (1 + \exp(r - \gamma))^{-1} \tag{6.2}$$

In this scenario, the argument r is set to the distance from the middle of the domain. For the sake of simplicity, the middle of the domain is set to the origin of the coordinate system, in turn yielding the definition of r:

$$r = \sqrt{x^2 + y^2} \tag{6.3}$$

The parameter γ specifies how far away from the center the water height falloff should happen. The height of the water volume defined by the sigmoid function is controlled by a factor β. As mentioned, a positive baseline water level is established with α, to which the sigmoid water volume is then added.

Combining those parameters, the initial water height is

$$h_0 = \alpha + \beta \cdot \sigma_\gamma(r) \tag{6.4}$$

This is demonstrated in figure 6.3a.

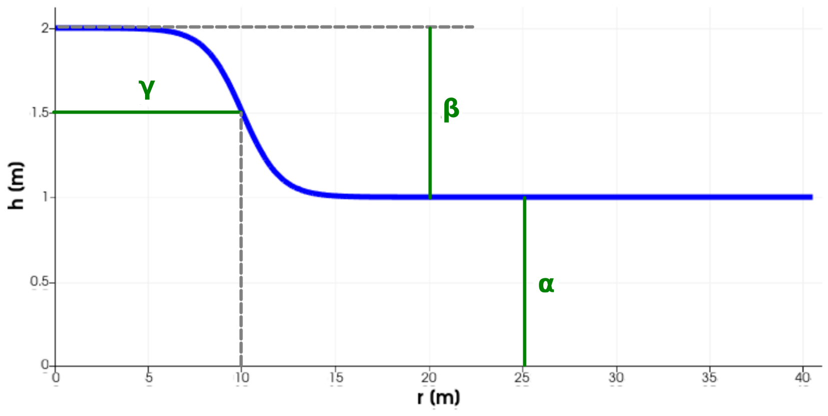
For this use case, the parameters are $\alpha = 1m$, $\beta = 1m$, and $\gamma = 10m$. The complete scenario definition is:

- ADER-DG solver
  - Order $\omega = 3$
- Domain width $w_{domain} = 81m$
- Subdivision depth $d = 4$
- Bathymetry $b = 0$
- Initial water height $h_0 = 1m + 1m \cdot \sigma_{10m}(r)$
- Wetness threshold $T_{wet} = 0.01m$

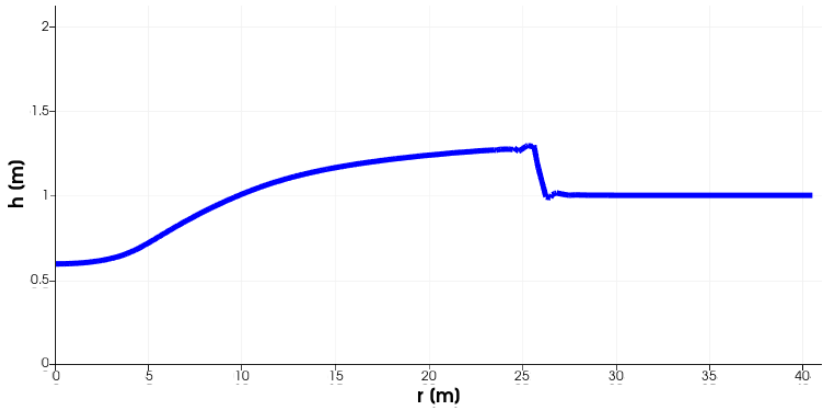
Figure 6.3a shows the cross-section of the domain's water volume through the origin with the smooth water falloffs from the sigmoid.

Observing the simulation during its computation, it is evident that it runs stably with a constant time step size.

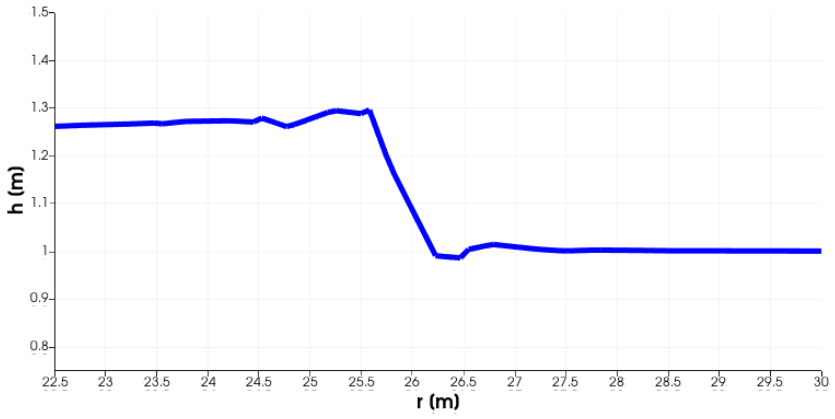
Initially, the water develops in a way that makes physical sense. It is pushing down on itself and escapes outwards from the origin. At some point, the wave takes the shape of a near-vertical edge, a so-called shock. In reality, the wave would break in the next moment. However, the shallow water equations do not consider friction and are thus incapable of simulating breaking waves. The result is that the wave keeps



(a) Initial water level defined by a sigmoid function.



(b) Water level after 4s producing a sharp edge with oscillations.



(c) Close up of the oscillations of figure 6.3b.

Figure 6.3.: Water level of the smooth water volume scenario plotted along its radius. Scenario parameters: ADER-DG order $\omega = 3$, $d = 4$.

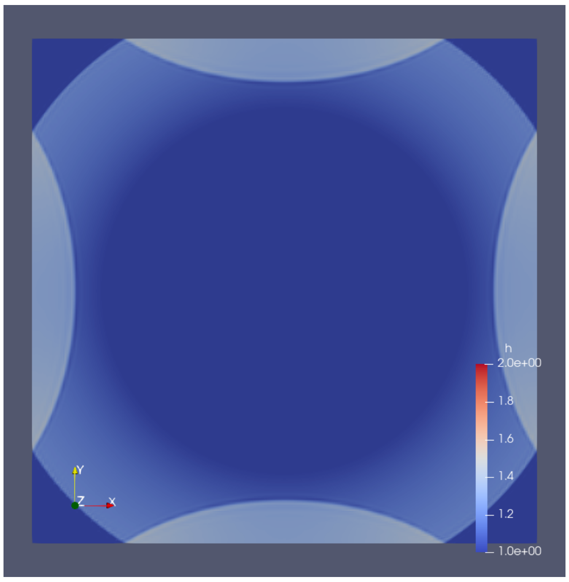


Figure 6.4.: A radial wave being reflected towards the domain middle at the boundary.

pushing the shock edge outward, as seen in figure 6.3b. It presents the simulation state at $t \approx 4s$ Additionally, there are oscillations of the water level at this edge caused by its polynomial representation, but they did not destabilize the simulation enough to break it. Figure 6.3c is a close-up of the shock edge, showing those oscillations.

This simulation also makes the effect of the reflecting boundary conditions visible, which have been explained in section 5.5. Figure 6.4 shows the radially outflowing wave being reflected into the domain after $t = 10$.

In conclusion, ExaHyPE 2, together with the SWE API, is capable of successfully simulating this scenario.

6.2.2. Dam break

The following experiment is another radial outflow scenario called a "dam break." This time, a sharply bounded water cylinder is placed on the resting lake. The noticeable feature of this cylinder is its vertical walls with a flat top, resulting in a discontinuity. The goal here is to analyze how ADER-DG handles such discontinuities.

Section 3.4 of article [**bib:leo**] already constructed and analyzed such a scenario. The simulation by Peano and the SWE API will be compared to those results. There, the base water level is at $\alpha = 4m$, and the cylinder is placed in the origin with a height of $\beta = 3m$ and a radius of $\gamma = 5m$, as demonstrated by figure 6.6a. Note that the figure plots the water level along the radius of the cylinder to stay consistent with the representation in [**bib:leo**].

A normalized water cylinder can be expressed as

$$c_\gamma(r) = \begin{cases} 1, & \text{if } r \leq \gamma \\ 0, & \text{otherwise} \end{cases} \tag{6.5}$$

And with that, the complete initial water height analogously to section 6.2.1:

$$h_0 = \alpha + \beta \cdot c_\gamma(r) \tag{6.6}$$

As for the previous experiments, here is the complete scenario definition:

- ADER-DG solver
  - Order $\omega = 7$
- Domain width $w_{domain} = 20m$
- Subdivision depth $d = 4$
- Bathymetry $b = 0$
- Initial water height $h_0 = 4m + 3m \cdot c_{5m}(r)$
- Wetness threshold $T_{wet} = 0.01m$

The reference, figure 3.6 of [**bib:leo**], shows the water level at $t = 0.3s$. That figure is adapted in figure 6.5 for a more straightforward comparison. Comparing this to figure the SWE API results in 6.6c, it is evident that the shapes of the water level match closely. This means that the shallow water equations implemented by the SWE API (see section 1) are well suited for accurate simulations.

While the sharp drop in the water level of the initial condition poses no problem for the ADER-DG solver, the resulting shock front produces oscillations like in section 6.2.1. They are visible in both figures 6.6c and 6.5.

One consistent way to reduce the oscillations is to reduce the ADER-DG order ω. Figure 6.6b shows the same simulation with an ADER-DG order of 3 instead of 7. The figure proves the reduction of the oscillations with a lower ADER-DG order. The downside of this, however, is that the accuracy is reduced.

Overall, this simulation scenario can be regarded as successful, further confirming the SWE API ADER-DG as a valuable tool for shallow water simulation.

6.3. Single wave on sloped beach

Now to a scenario, which is a combination of [**bib:swsb**] and [**bib:leo**]: The single wave run-up onto a sloped beach. This scenario is one step closer to reality than the previous

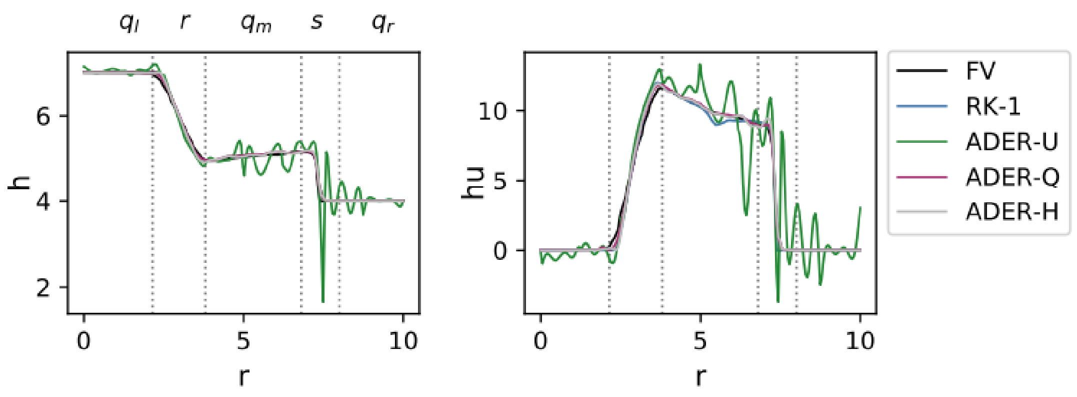


Figure 6.5.: Figure 3.6 of [**bib:leo**] showing their results of the dam break scenario.

one and is also a test for the dynamics of cells becoming dry and wet.

The sloped beach can be described with

$$b = -0.1x \tag{6.7}$$

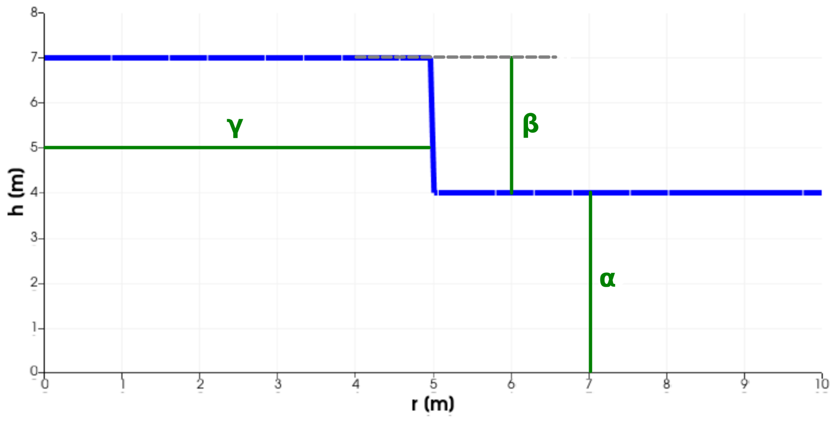
The initial wave shape is described by an equation similar to the one in section 4 of [**bib:swsb**], but with the parameters from equation 3.4 of [**bib:leo**]:

$$\begin{aligned} h_0 = \max\{0, 3m \cdot \exp(-0.4444 \cdot (\frac{x}{5000m} - 4.1209)^2) \\ - 9m \cdot \exp(-4 \cdot (\frac{x}{5000m} - 1.6384)^2) - b\} \end{aligned} \tag{6.8}$$

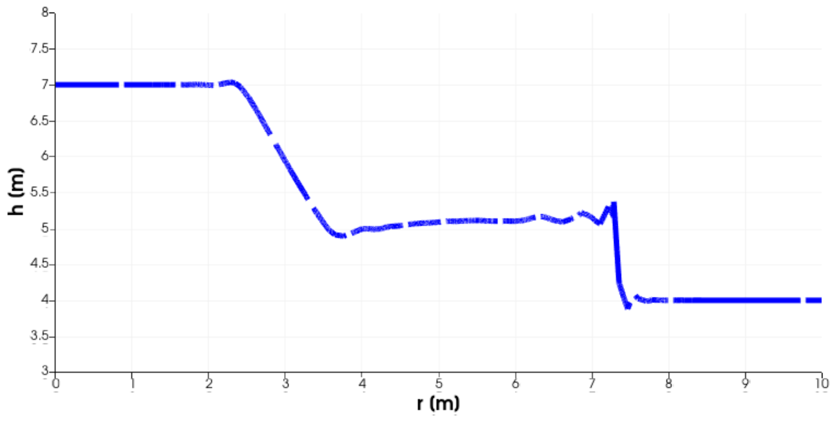
Figure 3.4 in [**bib:leo**] shows the wave development at $t_1 = 160s$, $t_2 = 175s$, and $t_3 = 220s$. The figure is also included here for simplicity in figure 6.7. The goal is to replicate these states with ExaHyPE2 and the SWE API. Note that their implementation uses a different SWE definition. Nevertheless, the overall shape of the plots should remain similar.

The complete scenario definition is

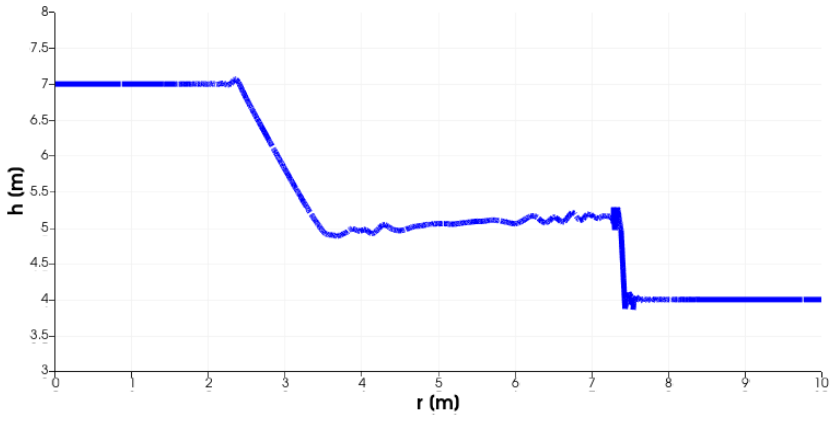
- ADER-DG solver
  - Order $\omega = 2$
  - Postprocessing: Non-negativity
- Domain width $w_{domain} = 50400m$
- Subdivision depth $d = 5$



(a) Initial water level of the radial dam break scenario, defined by a cylinder.



(b) ADER-DG order $\omega = 3$ simulation after $t = 0.3s$.



(c) ADER-DG order $\omega = 7$ simulation after $t = 0.3s$ with stronger oscillations compared to order $\omega = 3$.

Figure 6.6.: Water level of the radial dam break scenario plotted along its radius. Scenario parameter: $d = 4$.

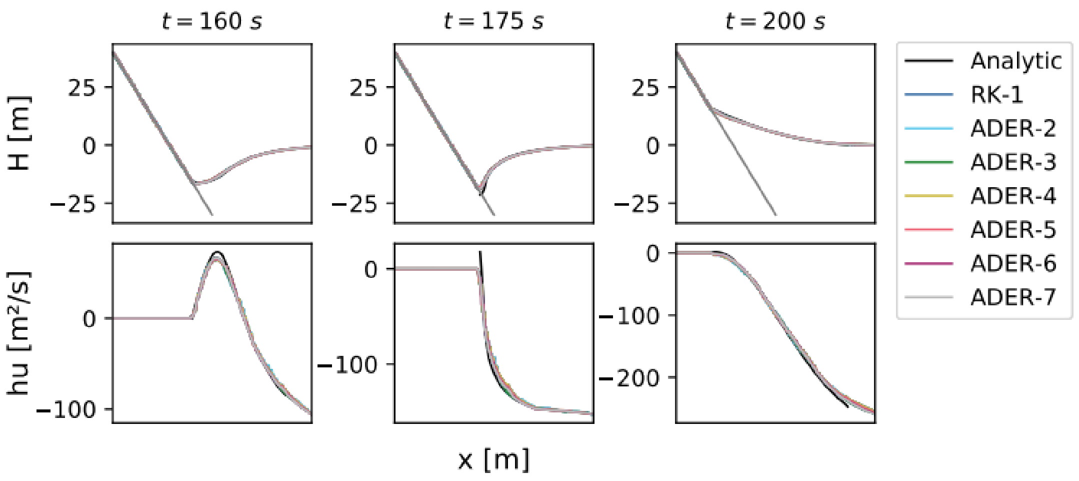


Figure 6.7.: Figure 3.4 of [**bib:leo**] showing their results for H and hv_x of the sloped beach scenario. Note that hv_x is denoted by hu in their figure. The original figure also featured plots for v_x, which were omitted here.

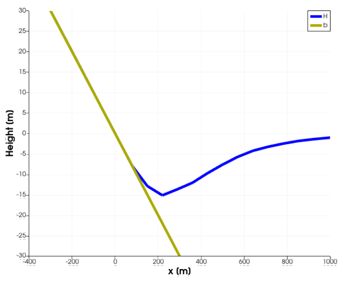
- Bathymetry $b = -0.1x$
- Initial water height: See equation 6.8
- Wetness threshold $T_{wet} = 0.01m$

Note that an ADER-DG order of $\omega = 2$ was used. This is because, for higher polynomial orders, NaN-values are produced before the final timestamp $t_3 = 220$. The NaNs are being generated in the fully dry corners of the domain, similarly to section 6.1.3. The difference to that scenario is, however, that the oscillations do not come from neighboring partially dry cells but instead from the cells outside of the boundary (see section 4.2.2). An order of 2 delays the NaN computation until after t_3, so the comparison to [**bib:leo**] can take place.

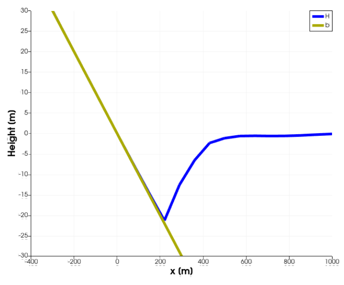
Figure 6.8 shows both the water levels $H = h + b$ as well as the velocities (times h) hv_x of the SWE API's results along the x direction at an arbitrary y coordinate in the same x interval. This representation is reasonable because the shape of the water and its temporal development is, by the scenario's design, independent of y.

It is evident that the shapes of both H and hv_x match closely to those of figure 6.7. However, there are visible differences in hv_x, at least for $t_1 = 160s$ and $t_2 = 175$, as seen in figures 6.8d and 6.8e. These differences, however, diminish in the further development of the water volume. Figure 6.8f shows minimal signs of them, concluding that they have negligible influence on the overall simulation quality.

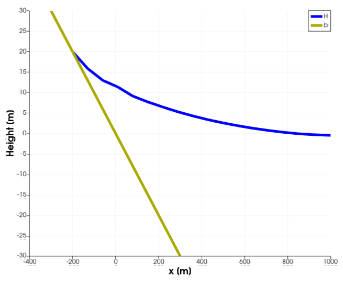
One interesting observation is that although this scenario features partially drying cells, the simulation did not break in those places like in section 6.1.3. This is because, in this



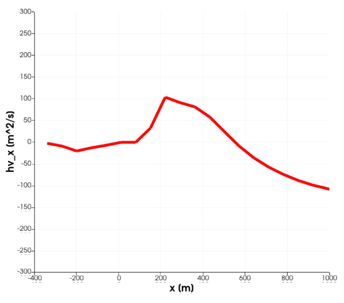
(a) b and H at $t_1 = 160s$.



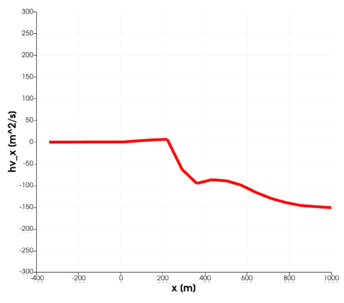
(b) b and H at $t_2 = 175s$.



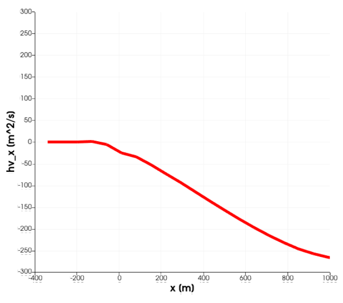
(c) b and H at $t_3 = 220s$.



(d) hv_x at $t_1 = 160s$.



(e) hv_x at $t_2 = 175s$.



(f) hv_x at $t_3 = 220s$.

Figure 6.8.: Simulation results of a wave run-up onto a sloping beach. H is shown in blue, b in green, and hv_x in red. Scenario parameters: ADER-DG order $\omega = 2$ with non-negativity postprocessing, $d = 5$.

scenario, there are no water-containing cells, where at least two neighbors are mostly dry.

The verdict of this scenario is that the SWE API can, in principle, accurately simulate the wave run-up onto a sloping beach with wetting and drying cells. However, fully dry domain corners are prone to instability, which may result in an early breaking of the simulation due to NaNs being computed.

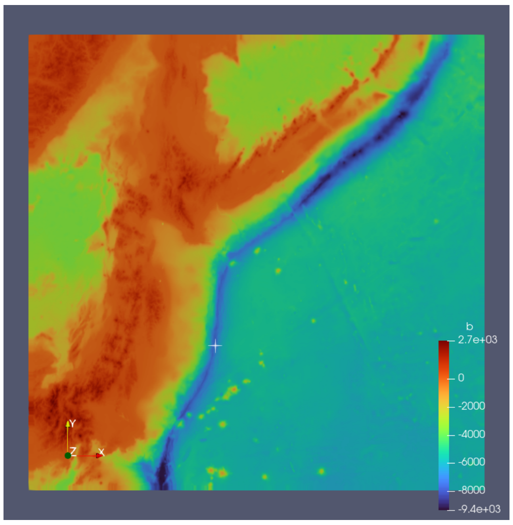
6.4. Real-world bathymetry: Japan

As mentioned in section I, the last scenario uses a real-world bathymetry of the Pacific Ocean east of the coast of Japan. It is inspired by the tsunami from 2011, which was partly responsible for the reactor meltdown at the Fukushima nuclear power plant and other severe devastation. For this experiment, another dam break is simulated, where a $\beta = 400m$ high and $\gamma = 100000m$ wide cylinder of water is placed on a base water level of $\alpha = 0$ to the east of Japan's land mass in the region of Tohoku. The bathymetry, extracted from a NetCDF file, is shown in figure 6.9a. This cylinder is much higher than the waves in 2011, with a maximum height of at most $40m$ [**bib:tohoku**]. This height was chosen for a clearer visualization of the results. Figure 6.9b shows the initial shape of the water. Japan's land mass sticking out of the water is also visible there. The goal of this scenario is to confirm that the SWE API can handle real-world bathymetry with all three types of cell states: Fully dry, fully wet, partially dry.

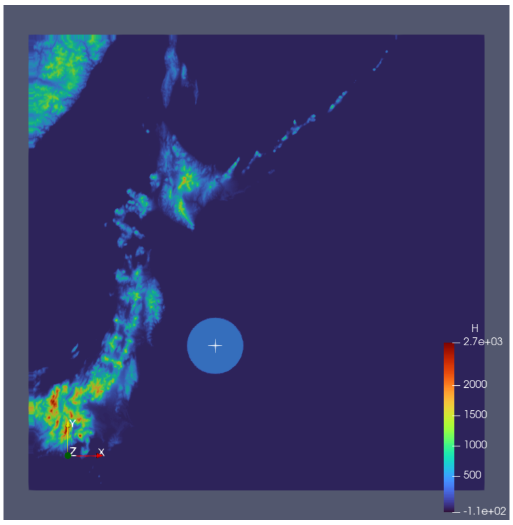
6.4.1. ADER-DG

The simulation with ADER-DG is set up to simulate $t_{max} = 5h = 18000s$ with the following configuration:

- ADER-DG solver
  - Order $\omega = 3$
  - Postprocessing: Non-negativity
- Domain width $w_{domain} = 3^{13} \cdot 1m = 1594323m$
- Subdivision depth $d = 5$
- Bathymetry b taken from the NetCDF file
- Initial water height $h_0 = \max\{0,\ 400m \cdot c_{100000m}(r) - b\}$ with $r = 0$ at coordinates $(150000m, 0m)$
- Wetness threshold $T_{wet} = 2.5m$
- Halo width $w_{halo} = 15000m$



(a) Bathymetry of and around Japan extracted from a NetCDF file.



(b) Initial water level of the scenario. The land mass of Japan is also visible.

Figure 6.9.: Initial configuration of the Tohoku tsunami scenario simulated with ADER-DG. The cross icon represents the center of the initial water cylinder. Scenario parameters: ADER-DG order $\omega = 3$ with non-negativity postprocessing, $d = 5$, $w_{halo} = 15000m$.

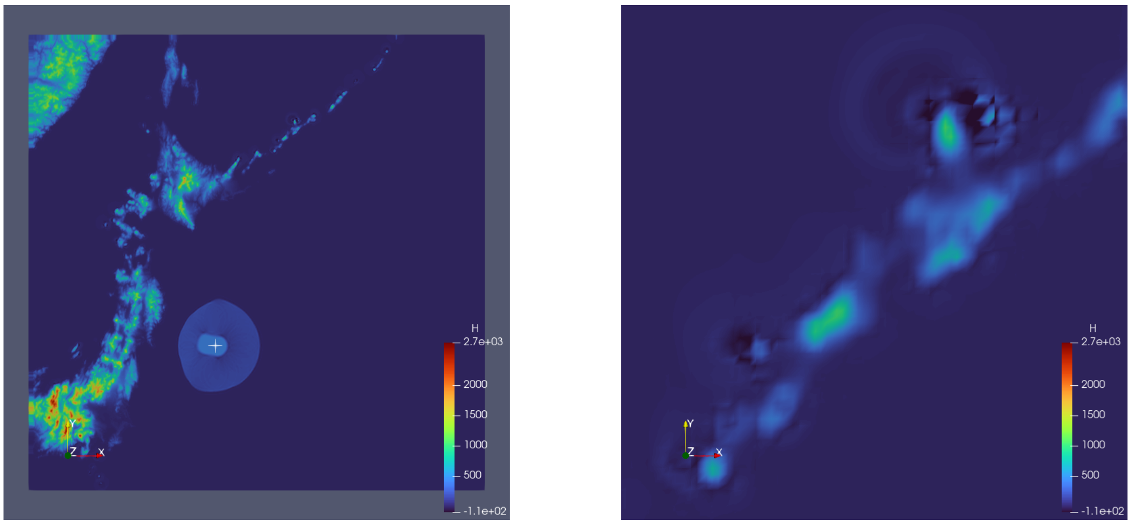
However, if no halo is defined, the simulation breaks at $t \approx 230s$ when NaNs start being calculated. This time, the NaNs are generated in the partially dry cells. This is due to the same oscillations, which were responsible for breaking the partially wet resting lake scenario in section 6.1.2. They are most prominent on the island of Iturup in the north-east of Japan. Adding a halo of width $w_{halo} = 15000m$ improves the simulation time to $t \approx 250s$, but the NaNs appear nonetheless. Figure 6.10b depicts the water level around the island, where the oscillations lead to NaNs in subsequent time steps.

The simulation is worth examining at its last valid time $t \approx 250s$. Figure 6.10a shows the entire domain. A visual inspection indicates a realistic development of the water volume, where the bathymetry in the coastal region deforms the radially outward-flowing water volume.

6.4.2. Finite Volumes

To further investigate the quality of the shallow water equations themselves, when implemented by the SWE API, the same scenario is simulated again with a finite volumes solver. The configuration is now:

- Finite volumes solver
    - Postprocessing: Non-negativity



(a) Full domain shows a realistic spreading of the wave before the simulation breaks.

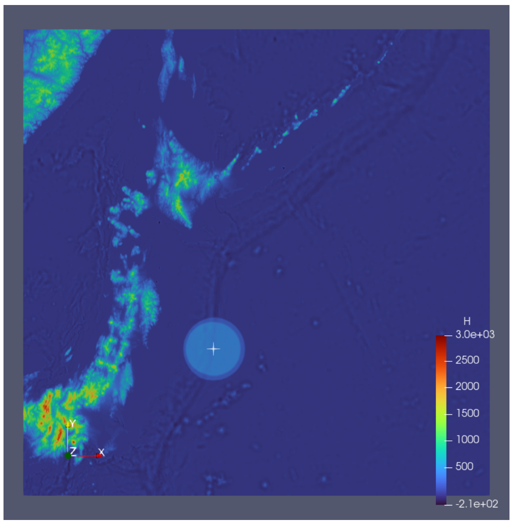
(b) Close-up of the island of Iturup, where the water level oscillations are severe enough to produce NaNs.

Figure 6.10.: Water level and land mass of the Tohoku tsunami scenario simulated with ADER-DG at $t \approx 250s$. Water level oscillations in partially dry cells lead to NaNs in subsequent time steps. Scenario parameters: ADER-DG order $\omega = 3$ with non-negativity postprocessing, $d = 5$, $w_{halo} = 15000m$.

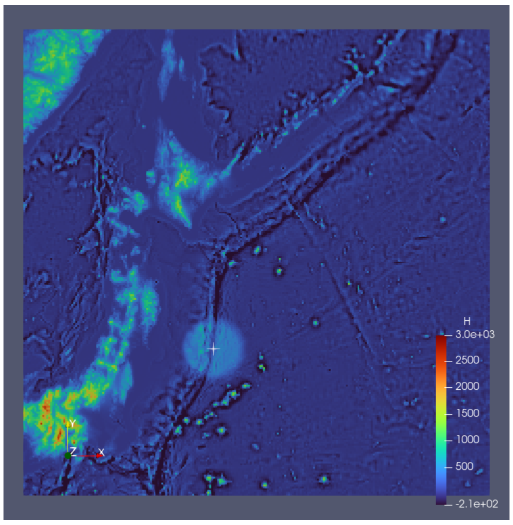
- Domain width $w_{domain} = 3^{13} \cdot 1m = 1594323m$
- Subdivision depth $d = 7$
- Bathymetry b taken from the NetCDF file
- Initial water height $h_0 = \max\{0,\ 400m \cdot c_{100000m}(r) - b\}$ with $r = 0$ at coordinates $(150000m, 0)$
- Wetness threshold $T_{wet} = 0.01m$
- Halo width $w_{halo} = 5000m$

Instead of ADER-DG, the simulation with the FV solver needs a narrower halo and runs stable for the whole duration until $t_{max} = 18000s$.

Figure 6.11a shows the simulation at $t \approx 30s$ shortly after the beginning. Since the water level (apart from the water cylinder) was defined as $H_0 = 0$, one would expect the water level to stay constant until the actual wave arrives. However, irregularities can be observed across the whole domain. Recalling the bathymetry (see figure 6.9a), it is evident that the irregularities happen where the bathymetry sharply changes. This is most prominent at the Japan Trench, the region of the Pacific Ocean where the Pacific plate moves under the Okhotsk Plate in a process called subduction. The Japan trench is



(a) $d = 7$.



(b) $d = 5$.

Figure 6.11.: Snapshots of the FV simulation at $t \approx 30s$ with different resolutions, i.e., domain subdivision depths d. Irregularities of the water level decrease with higher resolution. Scenario parameters: FV with non-negativity postprocessing, $d = 7$, $w_{halo} = 5000m$.

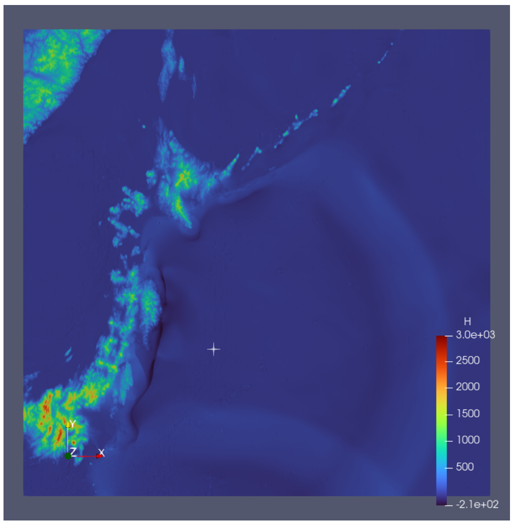
identifiable as the dark blue region in 6.9a.

The magnitude of the irregularities decreases with increasing resolution, i.e., increasing subdivision depth in the context of the SWE API. For comparison, figure 6.11b shows the water level of a simulation with a subdivision depth of $d = 5$, instead of 7, at the same time $t \approx 30s$. There, the irregularities are much more substantial.

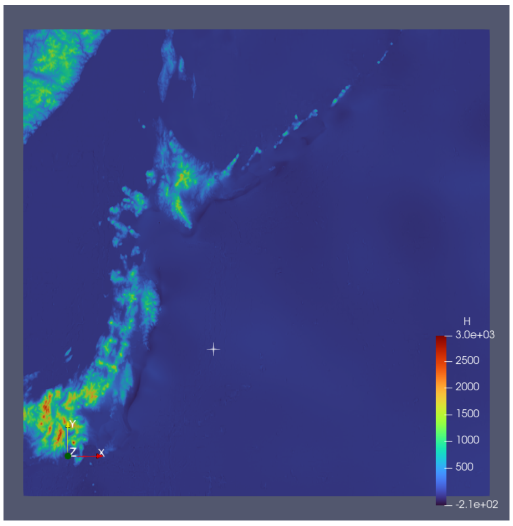
This phenomenon reduces the accuracy of the simulation significantly and is, therefore, a considerable disadvantage of the finite volumes method to ADER-DG, where it does not occur. However, this disadvantage is overshadowed by the complete breaking of the ADER-DG simulations, making the finite volumes method more feasible in such scenarios until a stable solution for partially dry cells is implemented.

Figure 6.12 shows more simulation snapshots. The snapshots further confirm the feasibility of SWE, as the water levels look physically and numerically plausible. The water level does not change much from $t \approx 12000s$ to $t_{max} = 18000s$ since the water is almost entirely at rest by then.

To conclude the last scenario, and thus the whole chapter, it can be said that ADER-DG offers higher accuracy and lower water level distortions caused by the bathymetry gradients compared to finite volumes. However, the breaking of the simulation at partially dry cells is a heavy disadvantage. Still, the SWE API generates plausible results, and the simulations run stably with the FV solver.



(a) $t \approx 3000$. The wave hits and encompasses Japan's land mass. Additionally, the wave is visibly reflected at the domain's bottom (south) border.



(b) $t \approx 12000s$. The wave almost fully dissipated, leading to the transformation of the simulation into a resting lake scenario.

Figure 6.12.: Snapshots of the FV simulation. Scenario parameters: FV with non-negativity postprocessing, $d = 7$, $w_{halo} = 5000m$.

7. Performance

In addition to the quality of the simulation results, the assessment of the runtime performance of the ADER-DG simulations is of interest.

To disambiguate the simulation time t from its actual runtime, the real-world time is denoted as τ. Note that the runtime performance is independent of the SWE API setup pipeline.

All following measurements were made on a consumer-grade computer with the following hardware specifications:

- CPU: AMD Ryzen 9 7950X with 16 cores at 4500MHz
- RAM: DDR5 at 6200MT/s

Peano 4 was configured to use OpenMP [**bib:openmp**] for multithreading. The times were measured over a runtime of $\tau_{max} = 600s$ per simulation. The simulation scenario used to generate the following results is similar to the Tohoku tsunami scenario from section 6.4:

- ADER-DG solver
  - Order $\omega = 3$
  - Postprocessing: Non-negativity
- Domain width $w_{domain} = 3^{13} \cdot 1m = 1594323m$
- Subdivision depth $d = 5$
- Bathymetry b taken from the NetCDF file
- Initial water height $h_0 = \max\{0,\ 400m \cdot c_{100000m}(r) - b\}$ with $r = 0$ at coordinates $(150000m, 0m)$
- Wetness threshold $T_{wet} = 2.5m$

7.1. Parallelization

The first important subject is the parallelizability of the solving process. As explained in section 4.1.1, the domain is traversed along the Peano curve, and segments of the curve can be mapped onto different CPU cores. This section analyzes the speedup when

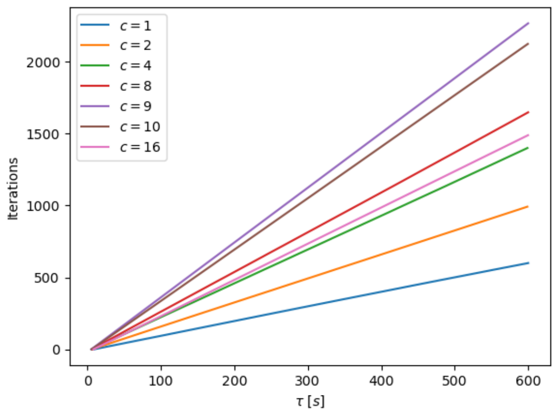


Figure 7.1.: ADER-DG iterations (time steps) computed in $\tau_{max} = 600s$ for different numbers of cores c.

using an increasing number of CPU cores. The number of cores c is controlled with the `OMP_NUM_THREADS` environment variable.

Figure 7.1 depicts, how many iterations, i.e. time steps, the simulation could compute in $\tau_{max} = 600s$. The different plots represent different numbers of cores c. It can be seen that the iterations for a set c take a constant time τ_{iter} to compute, which is reflected in the linearity of the plots. Another observation is that the setup time of the simulation, i.e., until iteration 0 commences, is $\tau_{setup} \approx 0$. Even though τ_{setup} includes the parsing of the NetCDF bathymetry files, as well as the setup of the cell mesh, and is in $\tau_{setup} \in \mathcal{O}(3^d)$ time, the speed of modern computers makes τ_{setup} negligible for a subdivision depth $d = 5$.

Table 7.1 lists the total number of iterations computed in $\tau_{max} = 600s$ and uses them to calculate the speedup of using multiple cores compared to a single one. From the table, it is evident that the parallelization does not yield linear speedup. The simulation speeds up until $c = 9$ cores and gets slower with $c > 9$. Note that this result may vary for different hardware configurations.

Overall, a benefit of parallelization is present, paving the way for efficient simulations.

7.2. Subdivision depth

Another prominent parameter that influences the runtime is the subdivision depth d.

Table 7.2 shows the setup times τ_{setup} and the average iteration times $\tilde{\tau}_{iter}$ calculated in $\tau_{max} = 600s$, depending on d. The measurements were made with the optimal number

c	Iterations	Speedup
1	601	1.0
2	993	1.65
3	1503	2.5
4	1401	2.33
5	1705	2.84
6	1712	2.85
7	1622	2.7
8	1649	2.74
9	2268	3.77
10	2125	3.54
11	1743	2.9
12	1579	2.63
13	1720	2.86
14	1558	2.59
15	1429	2.38
16	1490	2.48

Table 7.1.: Total ADER-DG iterations (time steps) computed in $\tau_{max} = 600s$ for different numbers of cores c and their speedup compared to $c = 1$.

d	τ_{setup}	$\tilde{\tau}_{iter}$
4	0.864	0.0399
5	5.909	0.266
6	47.232	2.13
7	419.246	18.9

Table 7.2.: Setup times τ_{setup} and average ADER-DG iteration times $\tilde{\tau}_{iter}$ for different subdivision depths d. The simulations ran for $\tau_{max} = 600s$.

of cores $c = 9$.

An exponential increase in both the setup times and the average iteration times can be observed. This makes sense, as the domain increases in size by $3^2 = 9$ for each increment of d. With higher values of d, a noticeable amount of time is spent on the cell mesh setup and the parsing of the bathymetry NetCDF file. While the $\tau_{setup} \approx 47s$ are still essentially negligible for $d = 6$, one has to take longer setup times into account for larger values of d. However, since simulations of high d need a significantly longer time to compute the desired number of iterations, τ_{setup} will remain only a tiny portion of the total simulation runtime.

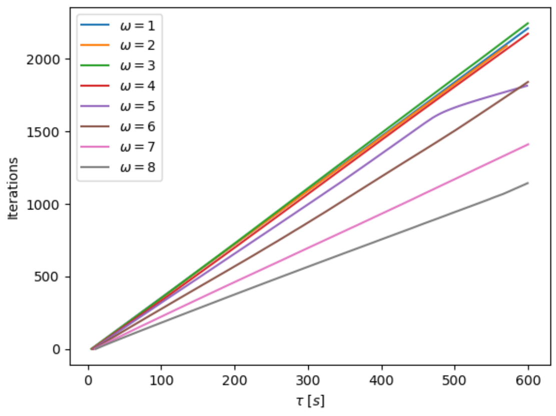


Figure 7.2.: ADER-DG iterations (time steps) computed in $\tau_{max} = 600s$ for different orders ω.

7.3. ADER-DG order

The last parameter which has an impact on the runtime is the ADER-DG order. An order of ω corresponds to $n = (\omega + 1)^2$ nodes per cell.

Figure 7.2 shows the number iterations processed in $\tau_{max} = 600s$. It can be seen that the iteration times stay nearly the same for $\omega \leq 4$. However, the simulation slows down for $\omega > 4$, reflected in less computed iterations in $\tau_{max} = 600s$. A possible reason is that this is due to the data load exceeding the capacity of the processor cache.

Another observation is the slowdown of the simulation with $\omega = 5$ and an early termination of the simulation for $\omega = 2$. These are the effects of the appearance of NaNs on the runtime. As soon as the NaNs cover the whole domain, the simulation terminates. This is why the $\omega = 2$ data does not extend to $\tau_{max} = 600s$. Until that happens, the NaNs must spread across the domain from their origin, in this case, the oscillation-prone region from figure 6.10b. This is slower compared to a stable runtime. While this slowdown is not noticeable for $\omega = 2$, due to the data still fitting inside the CPU cache, an increase of the iteration time τ_{iter} can be observed after the NaNs first start appearing for $\omega = 5$.

Part V.

Outlook and conclusion

As was demonstrated in this thesis, multiple weaknesses of ADER-DG exist. One may try to mitigate them in the future.

As seen in chapter 6, one of the biggest causes of instability was the handling of partially dry cells, which broke the simulations due to strong oscillations. In the future, this problem could be approached with different ideas. An example would be the so-called FV limiting, i.e., dynamic switching of the solver from ADER-DG to FV in problematic cells since it solves these regions stably.

To improve the interaction between adjacent cells, custom Riemann solvers can be implemented. Several types of Riemann solvers have been assessed in [**bib:ludwig**]. The SWE API offers a way to specify custom Riemann by modifying the appropriate solver template file.

Currently, the SWE API supports a fixed domain subdivision depth d. ExaHyPE 2 features so-called adaptive mesh refining (AMR). With AMR, subdomains can be further split or merged during runtime. When a solid AMR algorithm is chosen, it could be implemented in the centralized simulation logic collection of the SWE API or the scenario definitions.

In the current state, the SWE API implements reflecting boundary conditions. However, different and potentially more realistic options exist, for example, the so-called absorbing boundary conditions. While not mentioned in this thesis, they were tested during the development process. Unfortunately, they led to significantly less stable simulations. In the future, one might, therefore, try to implement different types of boundary conditions while finding a solution to preserve the stability of the simulations.

In this thesis, the only mentioned form of time step postprocessing was the enforcement of water height non-negativity. It is implemented in the SWE API by a generalized `Postprocessing` interface. This interface enables the addition of more complex postprocessing approaches in the future. For example, so-called slope limiting [**bib:slope_limiting**] promises to reduce oscillations within the ADER-DG cells. One might follow this idea by implementing such a slope limiter in the SWE API.

To summarize, this thesis explored the finite volumes and ADER-DG solvers in the context of shallow water equations. ADER-DG, in theory, promises high accuracy and an adequate overall simulation quality. However, the benchmarking results of part IV show that it does not entirely fulfill these promises. The instability of ADER-DG in dry regions make it, at least in its current implementation, inferior to FV. Further extensive research must be conducted to bring ADER-DG on par with FV.

The ExaHyPE 2 engine lays a solid foundation for developing better numerical solving approaches. The SWE API provides a convenient and easily extensible way to leverage this engine to its highest capacity, paving a smooth path for various simulation experiments. In the context of tsunamis, it aims to enhance the valuable simulation tools, improving the preparations and responses to real-world emergencies.

List of Figures