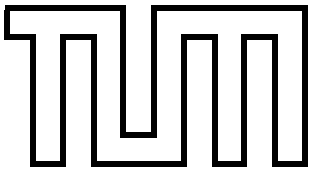


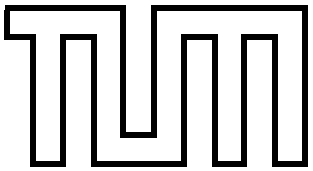
School of Computation, Information and Technology - Informatics

Technical University of Munich

Bachelor's Thesis in Informatics

Uncertainty Quantification Workflows Using the Shallow-Water Equations in ExaHyPE

Florian Wunderlich



School of Computation, Information and Technology - Informatics

Technical University of Munich

Bachelor's Thesis in Informatics

Uncertainty Quantification Workflows Using the Shallow-Water Equations in ExaHyPE

Arbeitsabläufe zur Unsicherheitsquantifizierung unter Verwendung der Flachwassergleichungen in ExaHyPE

Author:	Florian Wunderlich
Supervisor:	Prof. Dr. Michael Bader
Advisor:	Mario Wille, M.Sc.
Date:	15.09.2023

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

Munich, 15.09.2023 Florian Wunderlich

Acknowledgements

Many thanks to my advisor for providing me with this interesting opportunity for a Bachelor's thesis. I enjoyed our discussions on ExaHyPE 2 and past projects a lot, and wish to sincerely express my gratitude for the valuable input.

Abstract

Providing quantification measures for geohazardous events is of utmost importance not only for saving lives, but also for sustainable urban or industrial development in potentially vulnerable environments and many other fields in which geohazards play a role. This work implements an uncertainty quantification (UQ) workflow within the ExaHyPE 2 engine for the shallow-water equations (SWE). The SWE are a set of strictly hyperbolic partial differential equations, which are most commonly used in tsunami modeling. Therefore, an SWE application has been developed within ExaHyPE 2, and been verified for correctness. UQ has then been provided in the form of an UM-Bridge model server, which allows for running an SWE simulation with a displaced origin of the earthquake. The validity of this model server has also been verified. Additionally, an approach to guided adaptive mesh refinement (guided AMR) using the adjoint SWE developed by Davis and LeVeque has been implemented. The F-Wave solver mainly used in the SWE application is found to approximate the solution of the SWE reasonably well. But tsunami events failed to be solved correctly for reasons shown to be unrelated to the SWE application. The guided AMR has also been found to work to a certain degree, but here as well, the error is shown to not lie with the SWE application.

Contents

1. Introduction

Partial differential equations (PDEs) are at the core of modern scientific computing. They model relativity, sound, and fluids amongst many other natural phenomena. Solving those equations efficiently, meaning the results should be precise enough to be physically viable whilst keeping computation time under some constraint, is thus of utmost importance for not only advancing the related scientific fields, but also applying those results to real-world applications, which can change the lives of countless people. One such system of equations is the system of the shallow-water equations (SWE), which are used to model the flow of water for scenarios where the water depth is much smaller than the width of the wave. These most notably arise over the open ocean when undersea earthquakes displace hundreds of thousands of cubic meters of seawater, such as the Great Chilean Earthquake in 1960, or the 2011 Great East Japan Earthquake (Tōhoku earthquake). The resulting tsunami can devastate entire countries and have long-lasting consequences, both domestically and internationally. The 2011 Tōhoku tsunami not only led to the nuclear meltdown of three reactors at the Fukushima Daiichi Nuclear Power Plant, causing one of the worst nuclear disasters in the history of nuclear power, but also killed over 15,000 people [1].

While the Tōhoku tsunami caused a nuclear meltdown, it is far from being the deadliest, even when only looking at the young 21st century. The 2004 Sumatra earthquake and tsunami led to over 200,000 victims, directly affecting multiple countries in the Indian Ocean [2]. Advancements in understanding tsunami generation and spread can drastically improve early warning systems and protective measures. As a consequence of the 2004 Sumatra event, the German state initiated a cooperation with Indonesia to develop a tsunami early warning system (GITEWS) [3].

In 2018, the European Union has launched the Center of Excellence for Exascale in Solid Earth (ChEESE) to develop codes capable of using cutting-edge exascale computer capabilities to aid in understanding geohazardous events. And they also provide, among other things, probabilistic hazard assessment, and urgent computing capabilities for faster than real-time simulations of geophysical events. The first phase of ChEESE was concluded in 2022, and the second phase now began in 2023 [4]. Part of both phases of ChEESE is the development of an Exascale Hyperbolic PDE Engine (ExaHyPE, cf. [5]), which is being completely re-written for ChEESE-2P (ExaHyPE 2), as the first iteration was unable to improve its scalability during ChEESE-1P [4].

The object of this work is to establish an uncertainty quantification (UQ) workflow using the SWE in ExaHyPE 2. To achieve this, an SWE application capable of reliably and correctly solving the SWE will be implemented within the re-written ExaHyPE 2 engine. An application in ExaHyPE 2 encompassing a suitable implementation of the PDE as well as AMR code. Coupling to UQ codes will be provided by using the UQ and Model Bridge (UM-Bridge, cf. [6]) interface. Chapter 2 deals with ExaHyPE 2 and the SWE. The implementation of the SWE in ExaHyPE 2 is then explained in Chapter 3. Chapter 4 explores different approaches to adaptive mesh refinement (AMR). Chapter 5 addresses coupling with UQ codes using the UM-Bridge interface. In Chapter 6, qualitative and quantitative assessments of the delivered implementations are made. Chapter 7 provides a conclusion and an outlook on further work.

2. Related Work

2.1. ExaHyPE 2

ExaHyPE 2 (Exascale Hyperbolic PDE Engine) is an engine providing numerical methods for solving strictly hyperbolic PDEs for extreme-scale simulations [5]. To achieve extreme-scale capabilities, it utilizes both MPI and OpenMP [7] parallelization, as well as support for GPU offloading [8, 9, 10]. The mesh construction and dynamic AMR are handled by the Peano framework [11], which decomposes a given Cartesian grid into spacetrees based on the Peano curve. Peano is closely coupled with ExaHyPE 2 to ensure that ExaHyPE 2 can scale as best as possible for future exascale systems, such as the Frontier supercomputer at Oak Ridge National Laboratory[1].

Taken from [5], ExaHyPE 2 works on first-order hyperbolic PDEs:

$$\frac{\partial}{\partial t}Q + \nabla \cdot F(Q, \nabla Q) + B(Q) \cdot \nabla Q = S(Q) + \sum_{i=1}^{n_{ps}} \delta_i. \tag{2.1}$$

This system of equations is then discretized in space and time. ExaHyPE 2 provides general procedures for solving these equations at a given discrete time step on a discrete spatial mesh. The user then only has to specify the flux function $F(Q, \nabla Q)$, the source term $S(Q)$, and any non-conservative products $B(Q) \cdot \nabla Q$, as well as point sources δi. For any special treatment, a user-defined solving procedure can also be supplied as an alternative. Of course, boundary and initial conditions have to be supplied as well as. Eigenvalues have to be supplied for all methods that use adaptive time stepping.

One of the simplest ways of solving these PDEs is via a finite volume (FV) scheme with volumes Q_i^n, where n is the discrete time step, and i the volume index in one dimension. The discrete updates to the volumes can be applied by, for example, using a first-order Godunov scheme:

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x}(F^n_{i+\frac{1}{2}} - F^n_{i-\frac{1}{2}}). \tag{2.2}$$

The fluxes $F^n_{j+\frac{1}{2}}$ of the Riemann problem at the volume interfaces are then given by the chosen solver. For example, a local Lax-Friedrichs (Rusanov) solver:

$$F^n_{j+\frac{1}{2}} = \frac{F(Q_j^n) + F(Q_{j+1}^n)}{2} + \frac{S(Q_j^n) + S(Q_{j+1}^n)}{2} - \max\left(|\lambda_j^{max}|, |\lambda_{j+1}^{max}|\right)\frac{Q_{j+1}^n - Q_j^n}{2}. \tag{2.3}$$

The correct choice of solvers and schemes is vital for computation time and result. Higher-order time integration schemes such as a Runge-Kutta scheme improve error bounds and thus time integration accuracy. But they also take multiple evaluations of the SWE for different intermediate

[1] `https://www.ornl.gov/news/frontier-supercomputer-debuts-worlds-fastest-breaking-exascale-barrier`, accessed August 1st, 2023

results to calculate the $(n+1)$th time step. The choice of the solver on the other hand impacts the spatial accuracy, as the different solvers take varying degrees of information into account to calculate the interface fluxes. The given Rusanov solver only uses the interface-local maximum wave speed, hence the name *local* Lax-Friedrichs. Other approximate solvers, such as a Roe solver, also calculate using other wave speeds. Exact Riemann solvers even solve the Riemann problem posed at the interfaces using, for example, the Newton-Raphson method to calculate the middle state values Q^*.

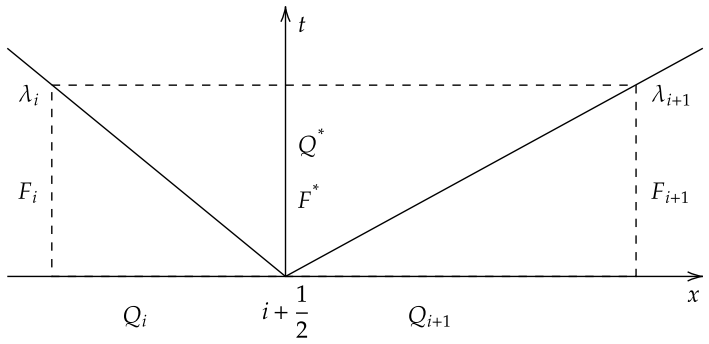


Figure 2.1.: Riemann problem in x-t plane at the state interface, the states are divided by the characteristic wave speeds. Figure modeled after [12].

In order to demonstrate the capabilities of the re-written ExaHyPE 2 engine, various applications have been implemented, such as astrophysical applications of modified gravity [13]. But the previous ExaHyPE iteration had many more applications, such as a whole plethora of UQ applications [14, 15], or cloud simulation [16]. As the general idea behind the ExaHyPE project has not changed with the iterations, the project aims to provide the same capabilities in the second iteration of ExaHyPE, that it had in its first iteration. To that end, this work aims to establish a new UQ workflow using the SWE within the re-written ExaHyPE 2 engine. Additionally, a guided AMR technique will also be implemented, which was available in the previous ExaHyPE iteration only for seismic simulations [17].

2.2. The Shallow-Water Equations

The SWE arise from a special case of depth-integrating the Navier-Stokes equations, wherein the horizontal scale of the domain is much greater than its vertical scale. This is the case for a great many bodies of water such as lakes, rivers, and oceans. In this work, the SWE are used to model various artificial test scenarios, with possible real-world applications being the simulation of the 2011 Tōhoku tsunami or the 2010 Chile tsunami.

The SWE used in this work can be written as:

$$\frac{\partial}{\partial t}\begin{pmatrix} h \\ hu \\ hv \end{pmatrix} + \frac{\partial}{\partial x}\begin{pmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{pmatrix} + \frac{\partial}{\partial y}\begin{pmatrix} hv \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{pmatrix} + \begin{pmatrix} 0 \\ ghb_x \\ ghb_y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \tag{2.4}$$

Here, $b(x, y)$ is the time-independent bathymetry measured in meters from a reference D, normally set at sea level with negative values lying below sea level, and positive values lying above sea level.

$h(x, y, t)$ is the total positive water depth measured in meters, and g is the earth's gravitational acceleration, which is set to 9.81m/s^2. The two velocities $u(x, y, t)$ and $v(x, y, t)$ are the velocities in m/s in the x- and y-dimensions respectively. Multiplied with $h(x, y, t)$, they form the momenta in the x- and y-dimensions $hu(x, y, t)$ and $hv(x, y, t)$. The variables b_x and b_y stand for the partial derivative of $b(x, y)$ in the given annotated dimension.

Following E. Toro [18], for the vector containing the conserved variables

$$Q = \begin{pmatrix} h \\ hu \\ hv \end{pmatrix}, \tag{2.5}$$

the following two flux functions can be immediately derived:

$$F(Q) = \begin{pmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{pmatrix}, \; G(Q) = \begin{pmatrix} hv \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{pmatrix}. \tag{2.6}$$

To account for the bottom topography in the form of the bathymetry, the source term is given as

$$S(Q) = \begin{pmatrix} 0 \\ ghb_x \\ ghb_y \end{pmatrix}. \tag{2.7}$$

The eigenvalues in the x- and y-dimensions are given by:

$$\lambda_{x_1} = u, \lambda_{x_{2,3}} = u \pm \sqrt{gh}, \tag{2.8}$$

$$\lambda_{y_1} = v, \lambda_{y_{2,3}} = v \pm \sqrt{gh}. \tag{2.9}$$

A detailed derivation of the SWE and their properties based on the laws of conversation of mass and momentum can also be found in [18]. A proof that the one-dimensional SWE are a set of strictly hyperbolic equations can be done in the following way:

1. Write the one-dimensional SWE in their primitive variable form:

$$\frac{\partial}{\partial t}\begin{pmatrix} h \\ u \end{pmatrix} + \frac{\partial}{\partial x}\underbrace{\begin{pmatrix} hu \\ \frac{1}{2}u^2 + gh \end{pmatrix}}_{=:f} \tag{2.10}$$

2. Formulate the Jacobian matrix J_f:

$$J_f(\begin{pmatrix} h & u \end{pmatrix}^T) = \begin{pmatrix} u & h \\ g & u \end{pmatrix} \tag{2.11}$$

3. The eigenvalues of J_f are found in Equation 2.8 as $\lambda_{x_{2,3}}$.
4. The system of a Jacobian $J \in \mathbb{R}^n \times \mathbb{R}^n$ is considered strictly hyperbolic if $\forall \alpha \in \mathbb{R} : \alpha J$ has n distinct real eigenvalues. For $\alpha = 1$, the eigenvalues are $\lambda_{x_{2,3}}$. With $h > 0$, $g = 9.81$, it holds that they are both distinct and real. For $\alpha \neq 1$, the eigenvalues scale with α. As $\alpha \in \mathbb{R}$, they always remain real. ■

A proof for the two-dimensional SWE can be found in [19]. It has to be noted that the SWE are only strictly hyperbolic as long as no wetting and drying is involved, as also noted in [19].

3. Implementation of the Shallow-Water Equations in ExaHyPE 2

First, the source term $S(Q)$ is not actually modeled as a source term in ExaHyPE 2. As $S(Q)$ requires b_x and b_y, it is better modeled as a non-conservative product $B(Q) \cdot \nabla Q$. The different components of the SWE are modeled as follows:

Algorithm 1: Flux Function

Input: q: state vector with elements h, hu, hv, b
$n \in \{x, y\}$: normal direction

Output: F: array containing flux in normal direction

Function `flux(`q`,`n`):`

- **if** $n = x$ **then**
  - $F \leftarrow \begin{pmatrix} hu & h(\frac{hu}{h})^2 + \frac{1}{2}gh^2 & hu\frac{hv}{h} \end{pmatrix}^T$
- **else**
  - $F \leftarrow \begin{pmatrix} hv & hv\frac{hu}{h} & h(\frac{hv}{h})^2 + \frac{1}{2}gh^2 \end{pmatrix}^T$
- **return** F

Algorithm 2: Non-Conservative Product

Input: q: state vector with elements h, hu, hv, b
∇q: gradient vector with elements $h_{grad}, hu_{grad}, hv_{grad}, b_{grad}$
$n \in \{x, y\}$: normal direction

Output: NCP: array containing non-conservative product in normal direction

Function `ncp(`q, ∇q, n`):`

- **if** $n = x$ **then**
  - $NCP \leftarrow \begin{pmatrix} 0.0 & ghb_{grad} & 0.0 \end{pmatrix}^T$
- **else**
  - $NCP \leftarrow \begin{pmatrix} 0.0 & 0.0 & ghb_{grad} \end{pmatrix}^T$
- **return** NCP

Algorithm 3: Eigenvalues

Input: q: state vector with elements h, hu, hv, b
$n \in \{x, y\}$: normal direction
Output: *Eigenvalues*: array containing eigenvalues in normal direction

Function `eigenvalues(`q`,`n`):`
  if $n = x$ **then**
    $u \leftarrow \frac{hu}{h}$
    $Eigenvalues \leftarrow \begin{pmatrix} u & u + \sqrt{gh} & u - \sqrt{gh} \end{pmatrix}^T$
  else
    $v \leftarrow \frac{hv}{h}$
    $Eigenvalues \leftarrow \begin{pmatrix} v & v + \sqrt{gh} & v - \sqrt{gh} \end{pmatrix}^T$
  return *Eigenvalues*

Another problem immediately arising from the SWE is the question on how to model wet-dry interfaces. As there is no water on dry land, the flux, the non-conservative product, and eigenvalues equate to 0. This is problematic for the Rusanov solver of Equation 2.3, as it still creates a flux onto dry land. As established before, the SWE lose their strictly hyperbolic properties when wetting and drying is involved. Thus, the physicality of this flux can not be guaranteed. Additionally, as the Lax-Friedrichs and Rusanov solvers are generally not well-balanced [20], they can create fluxes over a flat surface with varying water depth. Thus, an adaptation is required. For this purpose, the previously available feature of providing user-defined (FV) Riemann solvers has been re-introduced to ExaHyPE 2.

The FV solver interface of ExaHyPE 2 in general takes definitions for the flux, eigenvalues, source terms, non-conservative product, boundary conditions, and initial conditions as parameters. In the SWE application, the initial and boundary conditions are usually provided by the respective scenario. The application by default also uses a provided implementation of the flux function, the non-conservative product, and the eigenvalues (cf. Algorithms 1, 2, and 3). The user-defined FV Riemann solver implements the same interface as the predefined FV Rusanov solver, with the difference being that no implementations for the non-conservative product or the source term are given. As the user has direct control over the solution of the Riemann problem at a given cell interface, source terms and non-conservative products can directly be implemented. However, a flux function and a definition of the eigenvalues can still be provided so that they can be pre-computed. A time step computation of both, the user-defined solver, and the predefined Rusanov solver, is depicted as a UML activity diagram in Figure 3.1. An activity diagram of the whole SWE application can be found in Figure 3.2.

With control over the interface fluxes, a simple fix for dealing with wet-dry interfaces is introducing wall boundary conditions at these interfaces. The resulting fluxes are then used within an F-Wave solver [21], which provides the necessary well-balancedness. This solver was chosen as it is widely used in the "Clawpack" open source software package [22]. More elaborate solutions, such as exact Riemann solvers, should also be possible. An approximate wetting and drying solver from the previous iteration of ExaHyPE [23, 24] has been tried with the user-defined interface. However, it was found to not work as intended when confronted with wet-dry interfaces and real-world tsunami events. Therefore, it was discarded in favor of the aforementioned F-Wave solver.

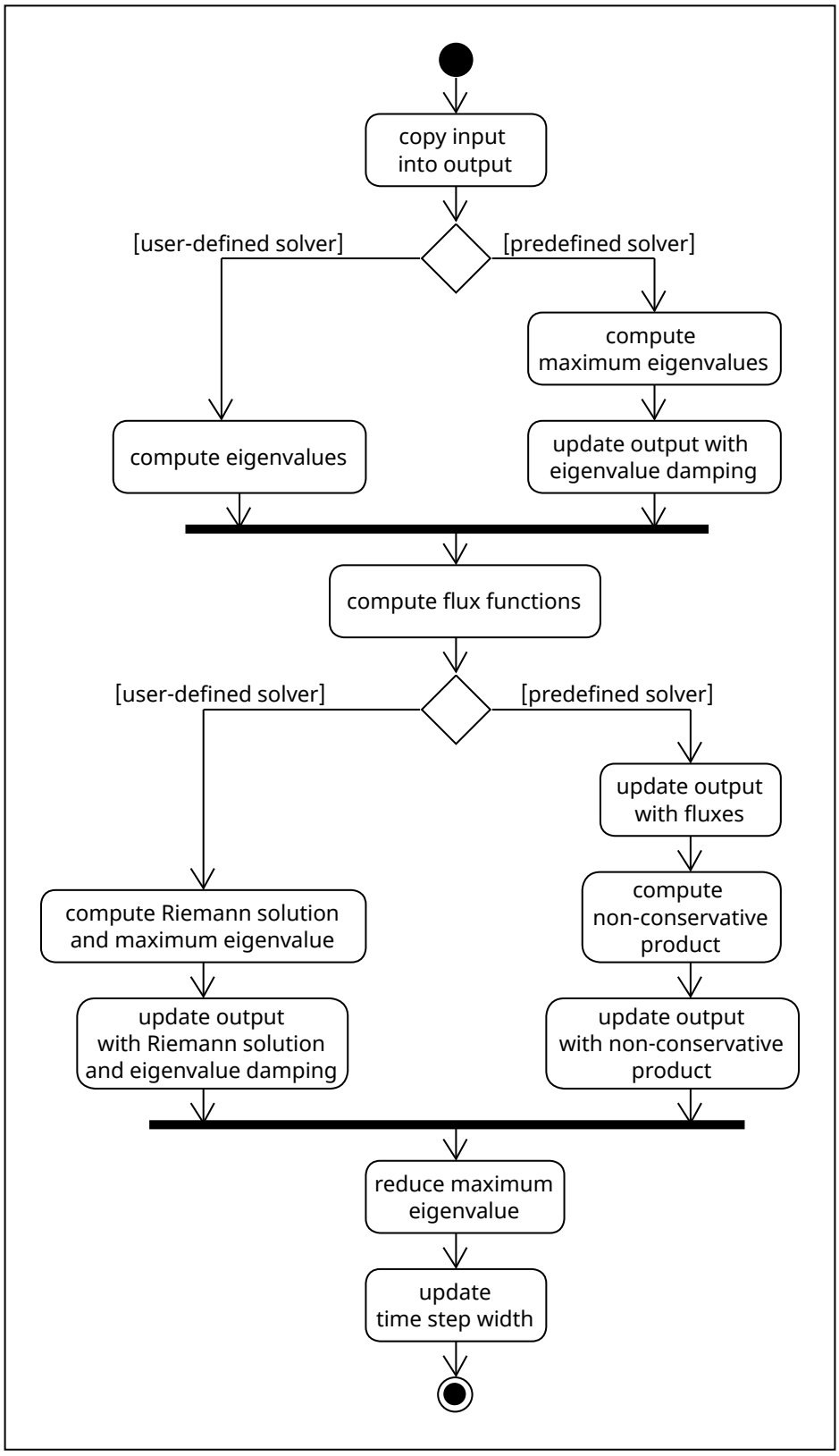


Figure 3.1.: UML activity diagram of an FV time step calculation.

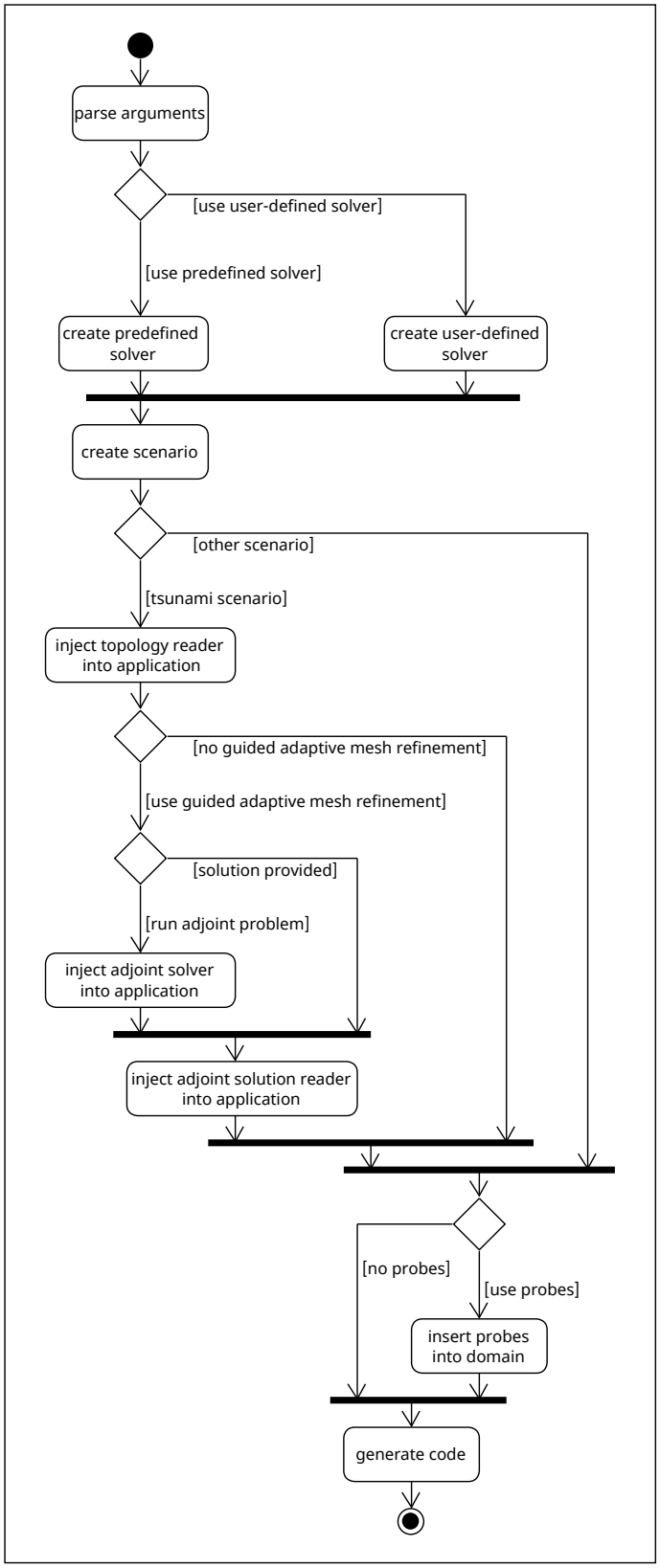


Figure 3.2.: UML activity diagram of the main SWE application.

4. Adaptive Mesh Refinement

To utilize the AMR capabilities of Peano as best as possible, two different refinement strategies have been taken into consideration for refining the volumes in a given simulation of the SWE.

4.1. Surface-Flagging

The first and most general strategy is the so-called "surface-flagging", wherein the free surface $\eta(x, y, t) = h(x, y, t) + b(x, y)$ is the sole criterion for refinement (cf. Algorithm 4). Together with a given tolerance, every volume in the simulation, whose free surface η is equal or larger than the tolerance, is to be refined. If the volume's free surface is below the tolerance, the volume is to be coarsened.

Algorithm 4: Surface-Flagging

Input: q: state vector with elements h, hu, hv, b
Output: refinement command
Data: $refinement\_tolerance$

```
1 Function refinement_criterion(q):
2     if h > 0 and abs(h + b) ≥ refinement_tolerance then
3         return refine
4     else
5         return coarsen
```

4.2. Guided Adaptive Mesh Refinement

The second strategy taken into consideration is guiding AMR in such a way, that only waves reaching a particular area of interest (AoI) are refined. In their paper [25], Davis and LeVeque showed that accuracy can be increased, and computational time can be decreased when using guided AMR for the SWE[1]. This guiding mechanism is realized by first solving the adjoint problem of the SWE over a given bathymetry. The simulation begins at a final time t_f, and solves the adjoint problem backwards in time. Then, the forward problem is solved on the same bathymetry, refining the mesh based on the inner product of the adjoint solution and the forward problem at certain timestamps (cf. Algorithms 5, 8).

[1] A more general approach can be found in [26]

Algorithm 5: Adjoint-Flagging

Input: t: time
x, y: volume center coordinates
q: state vector with elements h, hu, hv, b
Output: refinement command
Data: $refinement\_tolerance$

```
1 Function refinement_criterion(t, x, y, q):
2     if h > 0 and max_inner_product(t, x, y, q) ≥ refinement_tolerance then
3         return refine
4     else
5         return coarsen
```

The adjoint problem for the SWE is given in its primitive variable form as:

$$\frac{\partial}{\partial t}\begin{pmatrix}\tilde{\eta}\\ \tilde{\mu}\\ \tilde{\gamma}\end{pmatrix} + \frac{\partial}{\partial x}\begin{pmatrix}\tilde{\mu}\\ g\bar{h}\tilde{\eta}\\ 0\end{pmatrix} + \frac{\partial}{\partial y}\begin{pmatrix}\tilde{\gamma}\\ 0\\ g\bar{h}\tilde{\eta}\end{pmatrix} = \begin{pmatrix}0\\ 0\\ 0\end{pmatrix}. \tag{4.1}$$

Here, $\tilde{\eta}$ is the linearized free surface, $\tilde{\mu}$ and $\tilde{\gamma}$ the linearized wave speeds in the x- and y-dimension. The given variables are linearized about a sea at rest scenario with $\bar{\eta} = \bar{h}(x, y, 0) + b(x, y) = 0\text{m}$ and velocities $\bar{u}, \bar{v} = 0\text{m/s}$. The equations are solved in their conserved-variable form by a one-dimensional solver. Algorithm 6 is translated from the FORTRAN code provided in [25].

Algorithm 6: 1D Adjoint Solver

Input: q_l, q_r: 1D left and right state vectors with elements h, hu, b
Output: left and right going waves and maximum wave speed

Function `compute_net_updates(`q_l, q_r`):`
  $\hat{h}_l \leftarrow -b_l$, $\hat{h}_r \leftarrow -b_r$
  `// linearized wave speeds`
  $c_l \leftarrow$ `sqrt`$(9.81 \cdot \hat{h}_l)$, $c_r \leftarrow$ `sqrt`$(9.81 \cdot \hat{h}_r)$
  `// f-wave splitting`
  $\delta \leftarrow \begin{pmatrix} -hu_r \cdot c_r^2 + hu_l \cdot c_l^2 & (h_l + b_l) - (h_r + b_r) \end{pmatrix}^T$
  $\beta \leftarrow \begin{pmatrix} 1 & c_r \\ -1 & c_l \end{pmatrix} \cdot \delta$
  if $c_r + c_l \neq 0$ **then**
    $\beta \leftarrow \frac{1}{c_l + c_r} \cdot \beta$
  `// left going wave`
  $A^- \Delta Q \leftarrow \begin{pmatrix} c_l & 0 \\ 1 & 0 \end{pmatrix} \cdot \beta$
  `// right going wave`
  $A^+ \Delta Q \leftarrow \begin{pmatrix} 0 & -c_r \\ 0 & 1 \end{pmatrix} \cdot \beta$
  `// maximum wave speed`
  $\lambda_{max} \leftarrow$ `max(abs(`c_r`), abs(`c_l`))`
  return $A^- \Delta Q$, $A^+ \Delta Q$, and λ_{max}

The solver is deployed in a simple first-order dimensional splitting scheme (cf. Algorithm 7) [18, 27] on a static mesh:

$$Q^*_{i,j} = Q^n_{i,j} - \frac{\Delta t_x}{\Delta x}(A^- \Delta Q^n_{i+\frac{1}{2},j} + A^+ \Delta Q^n_{i-\frac{1}{2},j}), \tag{4.2}$$

$$Q^{n+1}_{i,j} = Q^*_{i,j} - \frac{\Delta t}{\Delta y}(B^- \Delta Q^*_{i,j+\frac{1}{2}} + B^+ \Delta Q^*_{i,j-\frac{1}{2}}). \tag{4.3}$$

Here, Δt_x is the maximum time step width resulting from the flux in the x-dimension. Δt is then $\max(\Delta t_x, \Delta t_y)$.

Algorithm 7: Dimensional-Splitting

Input: Q: 2D array of state vectors q
Output: maximum time step width
Data: $size_x$, $size_y$: number of volumes without boundary in x- and y-dimension
$A^-\Delta Q$, $A^+\Delta Q$, $B^-\Delta Q$, $B^+\Delta Q$: 2D arrays of update vectors
Δx, Δy: volume sizes, which are assumed to be identical

Function `compute_numerical_fluxes(`Q`):`
  `// x-sweep`
  $\lambda_x^{max} \leftarrow 0$
  for $i \leftarrow 1$ **to** $size_x + 1$ **do**
    for $j \leftarrow 0$ **to** $size_y + 1$ **do**
      $A^-\Delta Q_{i-1,j}$, $A^+\Delta Q_{i-1,j}$, $\lambda_{edge}^{max} \leftarrow$ `compute_net_updates(`$Q_{i-1,j}$, $Q_{i,j}$`)`
      $\lambda_x^{max} \leftarrow$ `max(`λ_x^{max}, λ_{edge}^{max}`)`
  $\Delta t \leftarrow 0.4 \cdot \Delta x / \lambda_x^{max}$
  for $i \leftarrow 1$ **to** $size_x$ **do**
    for $j \leftarrow 0$ **to** $size_y$ **do**
      $Q_{i,j} \leftarrow Q_{i,j} - \frac{\Delta t}{\Delta x}(A^+\Delta Q_{i-1,j} + A^-\Delta Q_{i,j})$
  `// y-sweep`
  $\lambda_y^{max} \leftarrow 0$
  for $j \leftarrow 1$ **to** $size_y + 1$ **do**
    for $i \leftarrow 1$ **to** $size_y + 1$ **do**
      $B^-\Delta Q_{i,j-1}$, $B^+\Delta Q_{i,j-1}$, $\lambda_{edge}^{max} \leftarrow$ `compute_net_updates(`$Q_{i,j-1}$, $Q_{i,j}$`)`
      $\lambda_y^{max} \leftarrow$ `max(`λ_{edge}^{max}, λ_y^{max}`)`
  $\lambda_{max} \leftarrow$ `max(`λ_x^{max}, λ_y^{max}`)`
  $\Delta t \leftarrow 0.4 \cdot \Delta x / \lambda_{max}$
  for $j \leftarrow 1$ **to** $size_y$ **do**
    for $i \leftarrow 1$ **to** $size_x$ **do**
      $Q_{i,j} \leftarrow Q_{i,j} - \frac{\Delta t}{\Delta x}(B^+\Delta Q_{i,j-1} + B^-\Delta Q_{i,j})$
  return Δt_{max}

According to equation 12.23 found in [28], the CFL condition in Algorithm 7 is set to

$$\frac{\Delta t}{\Delta x}\lambda_{max} < \frac{1}{2}. \tag{4.4}$$

The initial conditions of the adjoint problem are:

$$\tilde{\eta}(x, y, 0) = \begin{cases} 1.0\text{m} & (x, y) \in \text{AoI and } \bar{h}(x, y, 0) > 0.0\text{m} \\ 0.0\text{m} & \text{otherwise} \end{cases}, \tag{4.5}$$

$$\tilde{\mu} = \tilde{\gamma} = 0.0\text{m/s}. \tag{4.6}$$

It is then solved over the same bathymetry as the forward problem. The initial perturbation thereby does not stem from a displacement of the ocean floor, but rather from the AoI. More

specifically, only the already wet volumes within the AoI, as the adjoint solver is not capable of inundation. Thus, only the waves, which reach the AoI over water, will be refined in the forward problem.

Once the adjoint problem has been solved for the final time t_f, one commences with solving the forward problem. The mesh of the forward problem can be refined based on the maximum inner product

$$\max_{\bar{t}\in\{t_n,t_{n+1}\}} q^T_{adjoint}(\bar{x},\bar{y},t_f-\bar{t})q_{forward}(x,y,t), \tag{4.7}$$

where t_n, t_{n+1} are the timestamps of the adjoint solution, which frame the current time step t. $\bar{x}$ and $\bar{y}$ are coordinates of a volume of the adjoint solution. They are interpolated from the coordinates of the volume to be refined. The interpolation function of a coordinate z is given as

$$p_1(z, z_{max}, z^{min}_{adjoint}, z^{max}_{adjoint}) = \frac{z^{max}_{adjoint} - z^{min}_{adjoint}}{z_{max}} \cdot z + z^{min}_{adjoint}. \tag{4.8}$$

This function is defined for $z \in [0, z_{max}], z_{max} \in \{x_{max}, y_{max}\}$. The adjoint solution is limited in dimension z by $z^{min}_{adjoint}, z^{max}_{adjoint}$. Additional to the spatial AoI, a temporal AoI can also be specified with t_s. This leads to the maximum inner product to be calculated as

$$\max_{t_n\le\bar{t}\le t_{n+i}} q^T_{adjoint}(\bar{x},\bar{y},t_f-\bar{t})q_{forward}(x,y,t), \tag{4.9}$$

where t_{n+i} is the smallest timestamp of the adjoint greater than $t + (t_f - t_s)$. The adjoint solution is accessed in a time-reversed manner, as the adjoint problem is not actually solved backwards in time. It is rather solved forwards in time, and the solution is accessed backwards. Algorithm 8 has also been largely translated from the FORTRAN code provided in [25]. The exact algorithm used in ExaHyPE 2 is slightly altered, however. It does not compute the previous timestamp's inner product again, as it already has been compared to the maximum value. The previous inner product is therefore only necessary for the first index i, for which holds $timestamps_{i-1} \le t < timestamps_i$. This index can be precomputed, as it is only dependent on t and not on the volume. And since the first adjoint timestamp is always at $t = 0.0$s, this index i will always be larger than 0. Therefore, the variables $t_{previous}$ and $inner\_product_{max}$ can be initialized with $t_{previous} \leftarrow timestamps_{i-1}$, and $inner\_product_{max} \leftarrow$ `calculate_inner_product(`$i-1, x, y, q$`)`.

Algorithm 8: Calculation of Maximum Inner Product

Input: t: time
x, y: volume center coordinates
q: state vector with elements h, hu, hv, b
Output: $inner\_product_{max}$
Data: $timestamps$: 1D array of the timestamps of the adjoint solution
$Q^{adjoint}$: 3D array holding the time-dependent adjoint solution
t_s, t_f: temporal area of interest

```
1  Function max_inner_product(t, x, y, q):
2      t_previous ← 0
3      inner_product_max ← 0
4      for i ← 0 to |timestamps| − 1 do
5          if t_previous ≤ t + (t_f − t_s) and t < timestamps_i then
6              if i = 0 then
7                  inner_product_max ← calculate_inner_product(i, x, y, q)
8              else
9                  inner_product_max ← max(calculate_inner_product(i, x, y, q),
                       calculate_inner_product(i − 1, x, y, q), inner_product_max)
10             t_previous ← timestamps_i
11     return inner_product_max
12 Function calculate_inner_product(i, x, y, q):
13     if h = 0 then
14         return 0
15     p ← interpolate_values(i, x, y)
16     if h_p = 0 then
17         return 0
18     return (h + b) · (h_p + b_p) + hu · hu_p + hv · hv_p
19 Function interpolate_values(i, x, y):
20     x_a, y_a ← interpolate coordinates x and y onto Q_adjoint indices
       // access Q^adjoint in time-reversed manner
21     return Q^adjoint_{|timestamps|−i−1, y_a, x_a}
```

In their paper [25], LeVeque and Davis demonstrated for the hypothetical earthquake scenario AASZe04 of [29] the capabilities of their guided AMR approach. They found it to be slightly slower compared to a surface-flagging approach, which only refines the first wave hitting Crescent City, the AoI in this scenario. But when compared to a surface-flagging algorithm, which also refines the secondary wave until Crescent City, the guided AMR is found to be vastly faster. This work aims to replicate similar results using Peano's AMR capabilities as best as possible together with the guided AMR strategy.

5. Uncertainty Quantification Workflow with UM-Bridge

UM-Bridge (UQ and Model Bridge) is a software package designed to enable coupling of UQ codes with model codes [6]. Experiments conducted in the realm of UQ for the SWE usually are focused on tsunami simulations. There, the most common application is quantifying the tsunami source location [15] or inundation of specific areas [30]. Other parameters, whose uncertainty can be quantified, are for example the manning friction coefficient [31]. But UQ can also be used in the search for promising tidal stream energy systems [32]. As these simulations are computational intensive, they are best run on high performance systems.

UM-Bridge is specifically targeted for high performance computing environments [33], and has been in use with the previous iteration of ExaHyPE [14, 15]. Based on HTTP, it utilizes a client-server pattern. The client has to be provided for the UQ code. The server on the other hand has to be provided for the model code. The client then simply connects to the model server, requests a model evaluation for a specific input vector θ, and performs its operations on the returned output vector. A UML sequence diagram of this can be found in Figure 5.1.

The UM-Bridge model server developed as part of this work implements a very straight forward implementation of the forward model used to create the UQ for tsunami in [15]. The Python version of UM-Bridge requires the user to specify the input and output vector sizes, a `__call__()` method that evaluates the model, and, in this case, the `supports_evaluate()` method to return `True`. As the forward model is set up in the same fashion as the one used by Seelinger et al. in [15], the input vector is expected to contain two elements, the x and y coordinate of the displacement's center. The output on the other hand is variable in size, being double the number of probes to be inserted into the domain, as each probe returns a maximum water height and the corresponding time of the measurement. The server could also easily be adjusted to return other parameters.

When a model evaluation is requested by the client and an appropriate `config` dictionary supplied, the SWE application is compiled using the options provided in `config`. The displacement's origin does not have to be stated in `config`, but rather be passed as the input vector θ, which is automatically added to the compile options. After a successful compile, the application is automatically run. If guided AMR is used and no precomputed solution file supplied, then a simulation of the adjoint problem will be run first, before running the forward problem's simulation. The server then returns the maximum water height and the corresponding time measurement for each probe as a vector. This behavior could be changed to accommodate for quantities of interest different from the maximum water height.

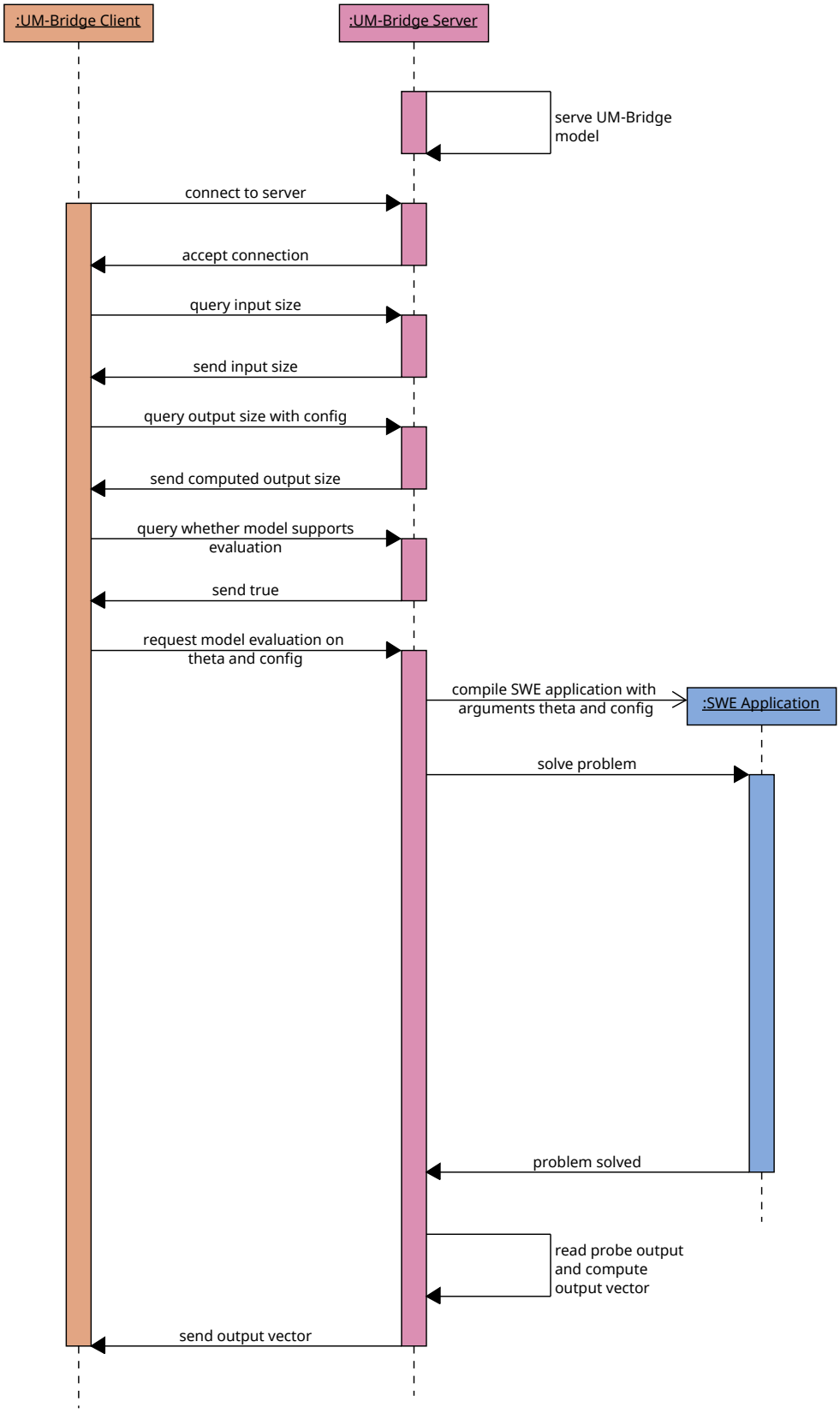


Figure 5.1.: UQ workflow using UM-Bridge.

6. Results

To validate the SWE implemented in ExaHyPE 2, a number of artificial scenarios are used. By default, ExaHyPE 2 supports a Rusanov FV solver, which is taken to deliver correct approximate results, to which the F-Wave solver is compared to. Real world applications are tested using the 2011 Tōhoku tsunami. In general, scenarios are run on a $[0, 10]\text{m} \times [0, 10]\text{m}$ domain with a configured volume size of 0.1m using a time step relaxation (TSR) factor of 0.4 and no AMR. For quantitative comparisons, the values are plotted along a line from $(5, 0)\text{m}$ to $(5, 10)\text{m}$. For scenarios involving wet-dry interfaces, boundary conditions are applied to those volumes where $h(x, y, t) \leq 10^{-2}\text{m}$[1].

6.1. Radial Dam Break

The radial dam break scenario is ideal to test the flux implementation for its correctness. In this scenario, the non-conservative product equates to zero, and all initial flux stems from the hydrostatic pressure $\frac{1}{2}gh^2$. The initial conditions are given for $(x, y) \in [0, N]\text{m} \times [0, N]\text{m}$ as:

$$h(x, y, 0) = \begin{cases} 1.1\text{m} & \text{if } \sqrt{(\frac{N}{2} - x)^2 + (\frac{N}{2} - y)^2} \leq \frac{N}{10} \\ 1.0\text{m} & \text{otherwise} \end{cases}, \tag{6.1}$$

$$hu(x, y, 0) = hv(x, y, 0) = 0.0\text{m}^2/\text{s}, \tag{6.2}$$

$$b(x, y) = 0.0\text{m}. \tag{6.3}$$

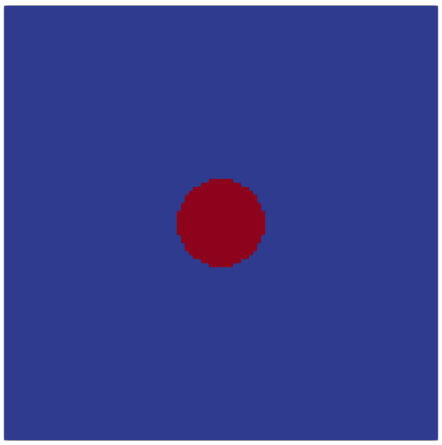


Figure 6.1.: Initial water height of radial dam break scenario. Red = 1.1m, blue = 1.0m.

[1] A detailed artifact description can be found under `https://doi.org/10.5281/zenodo.8305725`.

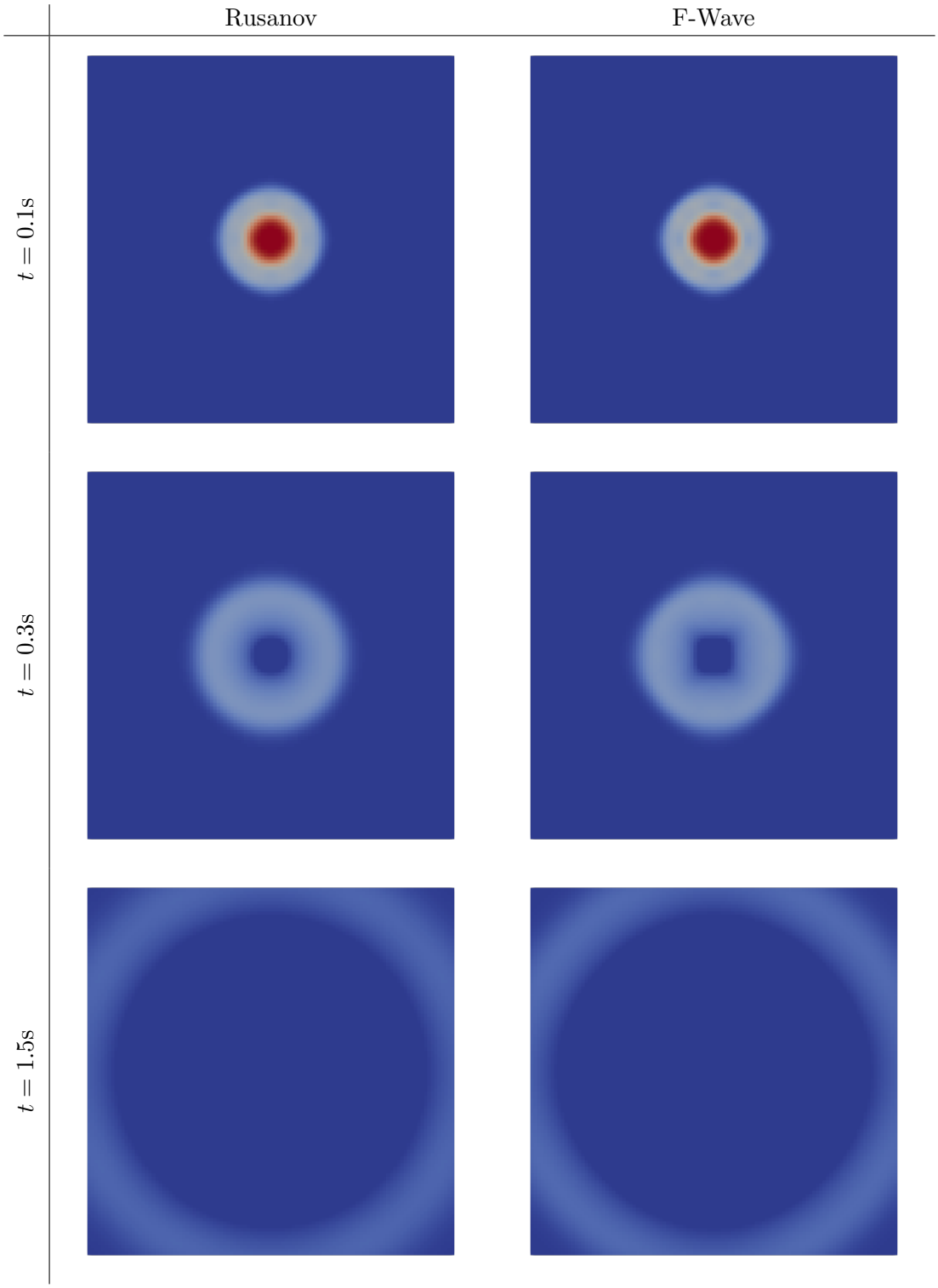


Figure 6.2.: Qualitative comparison of h in the radial dam break scenario between Rusanov and F-Wave solver. Colors range from red = 1.1m to blue = 1.0m.

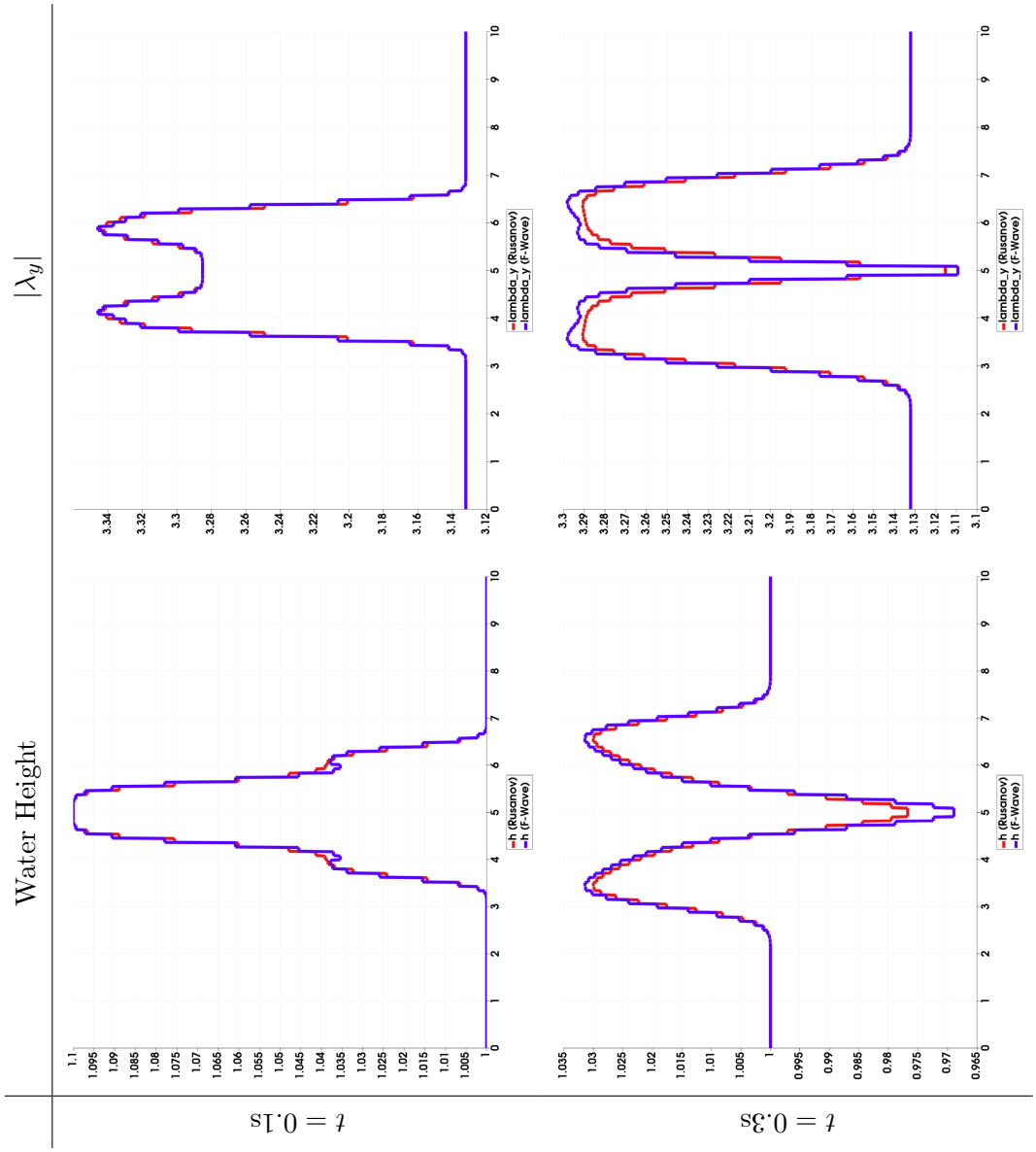
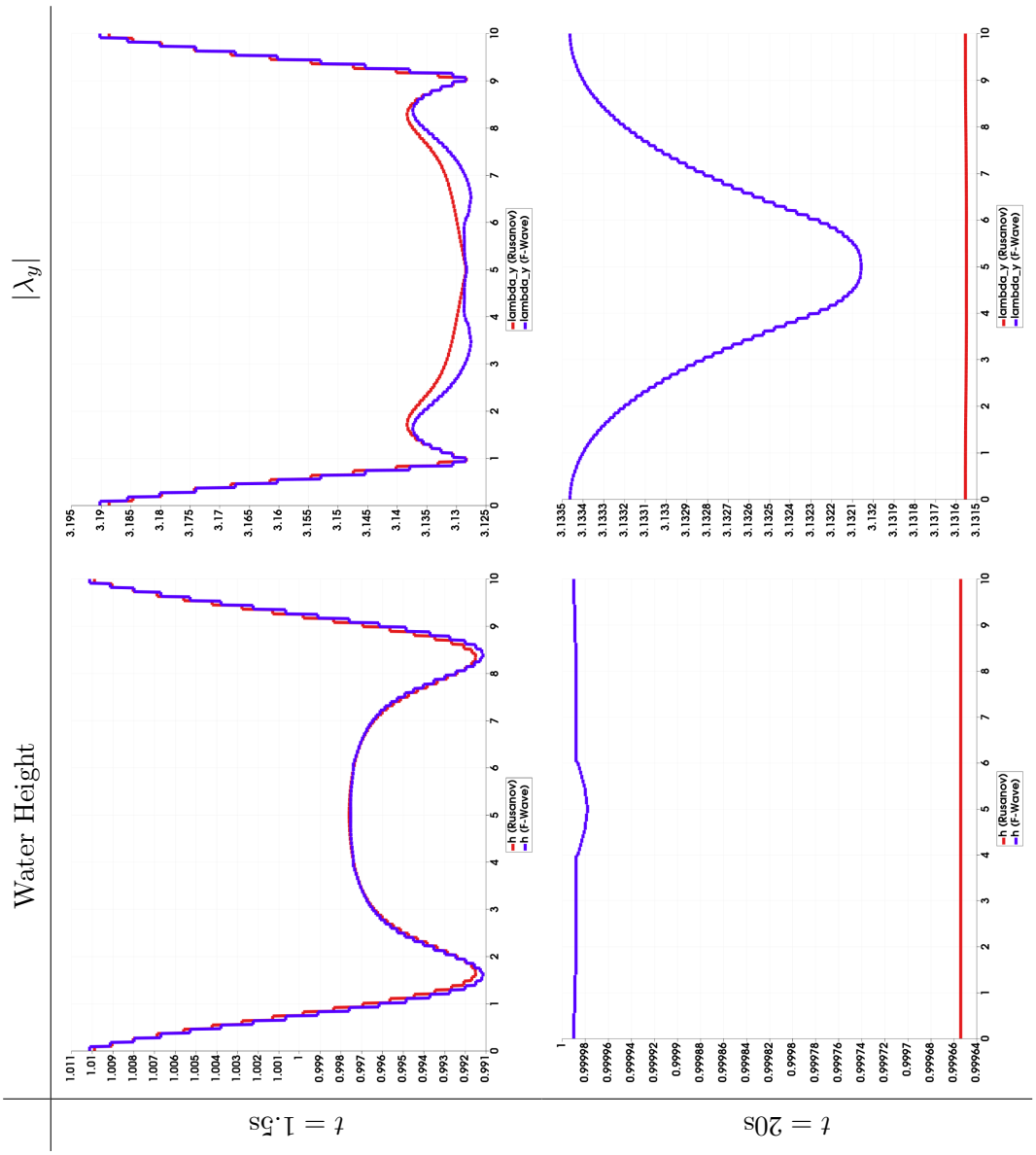


Figure 6.3.: Quantitative comparison of h and $|\lambda_y|$ in the radial dam break scenario between Rusanov and F-Wave solver. The quantities are plotted over the simulated time for $t \in [0.1, 20]$s (continued on the next page).


Water Height
$|\lambda_y|$
$t = 1.5$s
$t = 20$s
h (Rusanov)
h (F-Wave)
lambda_y (Rusanov)
lambda_y (F-Wave)

When looking at the qualitative comparison between the F-Wave solver and the Rusanov solver in Figure 6.2, the results look similar. The quantitative comparison in Figure 6.3 also supports this finding. However, the F-Wave solver and Rusanov solver disagree over the limit for the water height. The F-Wave solver almost reaches the expected limit of $h = 1.0\text{m}$ whilst maintaining a noticeable dip where the initial conditions were set to $h = 1.1\text{m}$. This dip is consequentially present within the wave speeds, as the wave speeds reduce to $\sqrt{gh}$. As the F-Wave solver is just a more elaborate approximate Riemann solver, some artifacts are to be expected. The Rusanov solver on the other hand reaches a constant limit, albeit that this limit is lower than the expected limit, and also lower than the limit of the F-Wave solver. This is also reflected in the wave speeds, as the Rusanov solver's $|\lambda_y|$ reaches a constant limit less than the expected $\sqrt{gh} \approx 3.132\text{m/s}$. It can be concluded that both the F-Wave solver and the Rusanov solver deliver reasonable results in terms of flux and eigenvalues. However, the F-Wave solver delivers quantitatively and qualitatively better results than the Rusanov solver.

6.2. Radial Bathymetry Dam Break

After verifying that the flux and eigenvalue functions works correctly, the non-conservative product can be verified next. For this, a new radial dam break scenario is devised:

$$h(x, y, 0) = 1.0\text{m}, \tag{6.4}$$

$$hu(x, y, 0) = hv(x, y, 0) = 0.0\text{m}^2/\text{s}, \tag{6.5}$$

$$b(x, y) = \begin{cases} -0.9\text{m} & \text{if } \sqrt{(\frac{N}{2} - x)^2 + (\frac{N}{2} - y)^2} \leq \frac{N}{10} \\ -1.0\text{m} & \text{otherwise} \end{cases}. \tag{6.6}$$

Once again, $(x, y) \in [0, N] \times [0, N]$ in meters. As the flux function results in the same values over the whole domain, this scenario's initial flux is only derived from the non-conservative product ghb_x, ghb_y. Since the difference in the free surface η is the same as in the prior radial dam break scenario (cf. Figures 6.4, 6.1), the scenario is also expected to behave similarly.

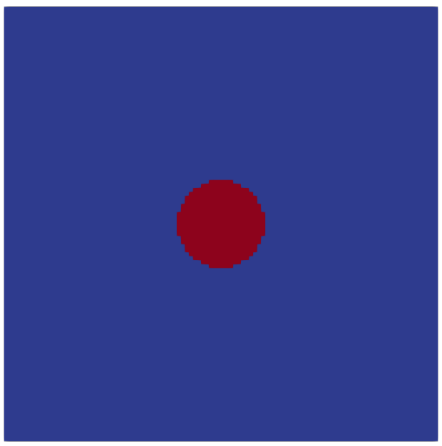


Figure 6.4.: Initial free surface η of the radial bathymetry dam break scenario. Red corresponds to 0.1m whilst blue corresponds to 0.0m.

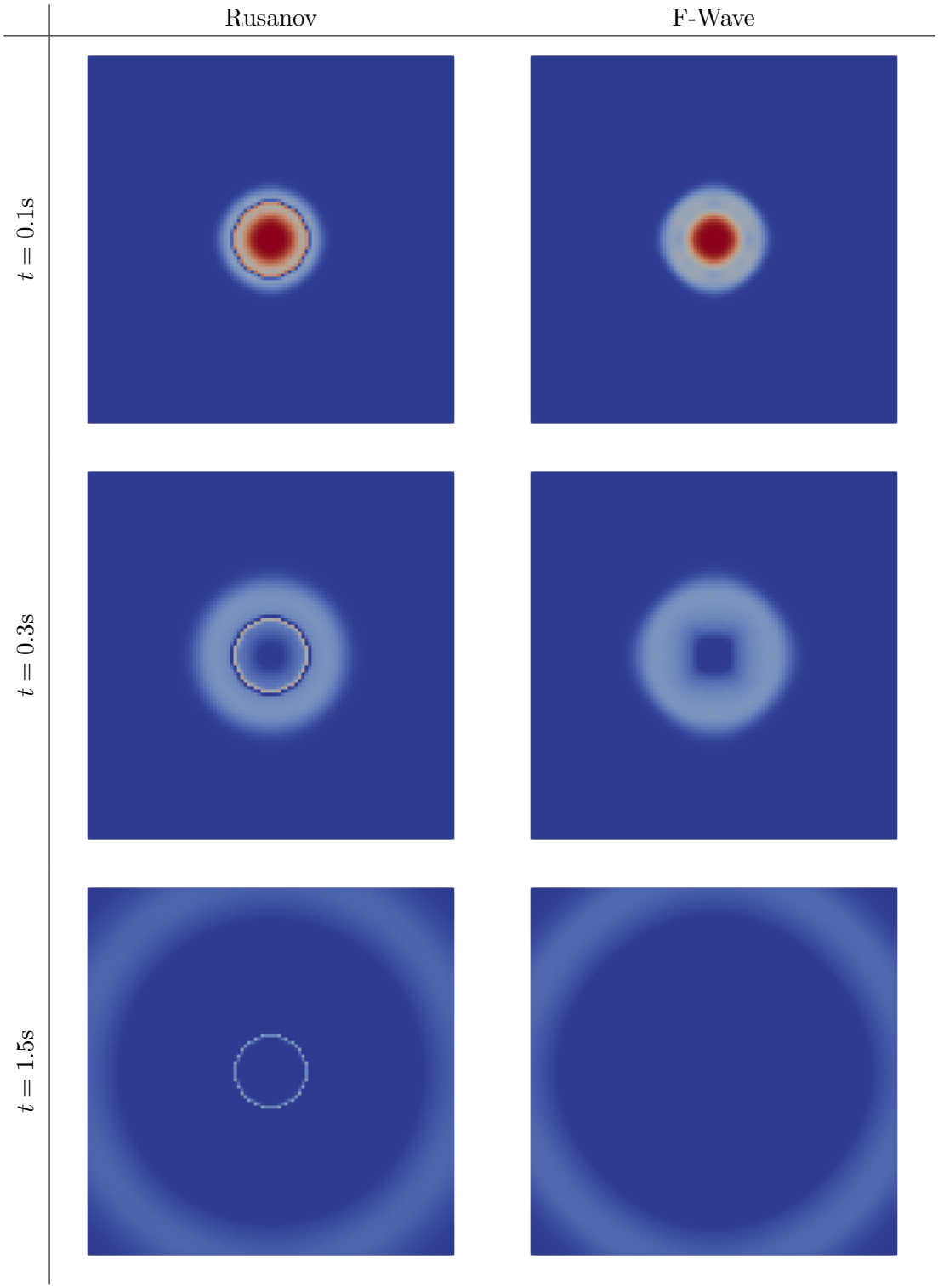


Figure 6.5.: Qualitative comparison of η in the radial bathymetry dam break scenario between Rusanov and F-Wave solver. Colors range from red = 0.1m to blue = 0.0m.

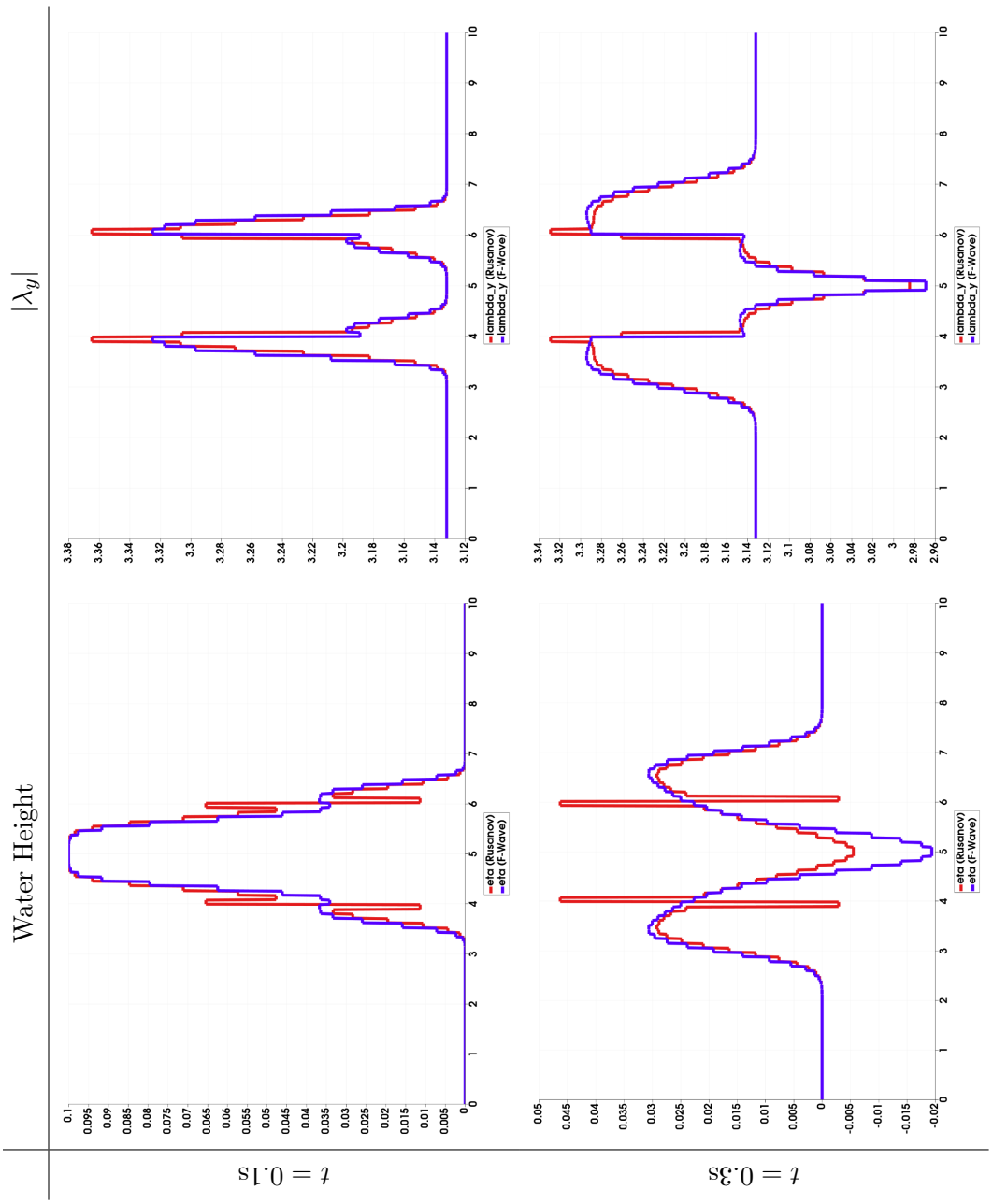
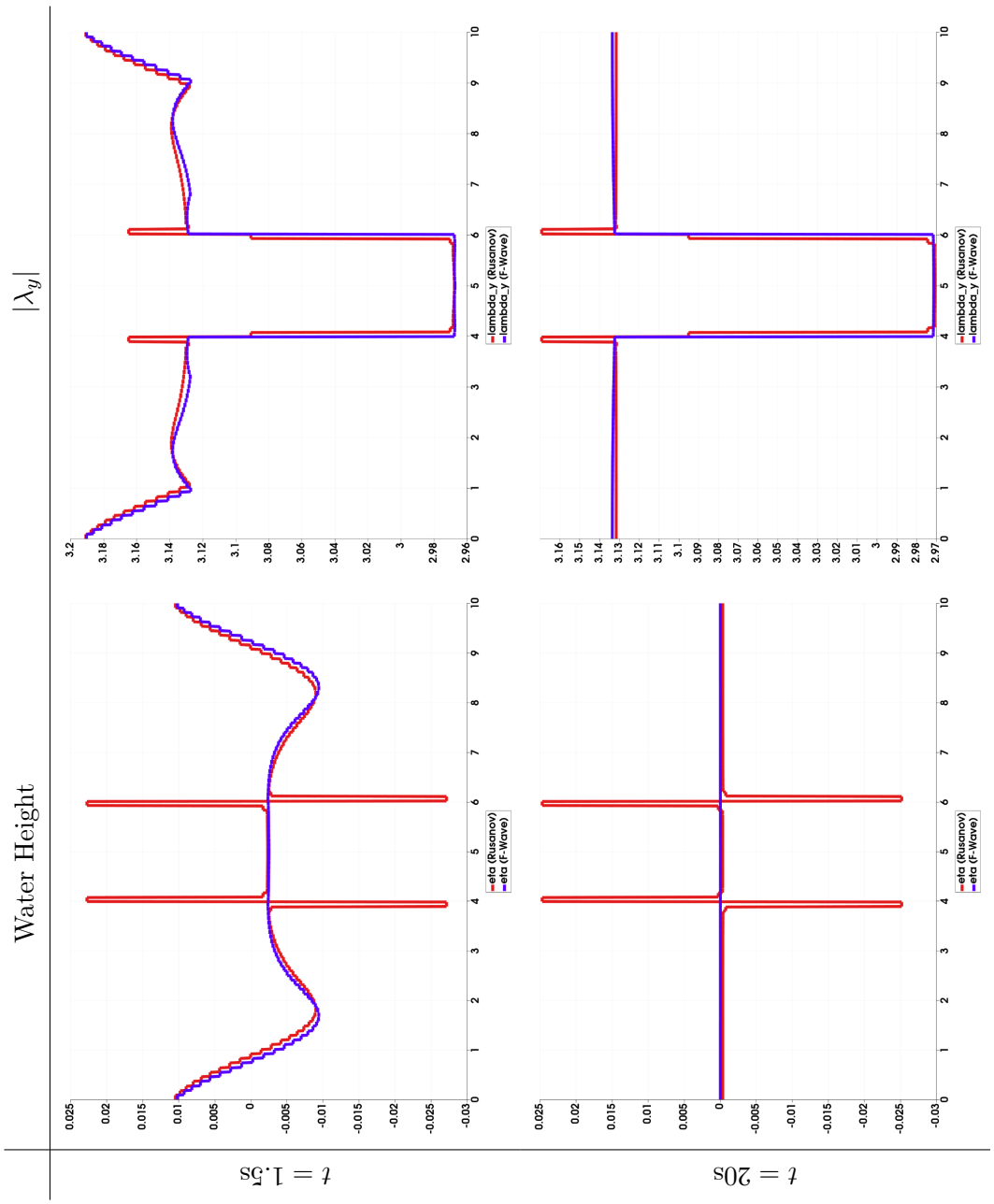


Figure 6.6.: Quantitative comparison of η and $|\lambda_y|$ in the radial bathymetry dam break scenario between Rusanov and F-Wave solver. The quantities are plotted over the simulated time for $t \in [0.1, 20]$s (continued on the next page).


Water Height
$|\lambda_y|$
$t = 1.5$s
$t = 20$s
eta (Rusanov)
eta (F-Wave)
lambda_y (Rusanov)
lambda_y (F-Wave)

As can be seen in Figure 6.5, the F-Wave solver produces a highly similar solution to the radial dam break scenario. The Rusanov solver on the other hand shows a noticeable artifact at the discontinuity between $b = -1.0\text{m}$ and $b = -0.9\text{m}$. This artifact persists for the whole duration of the simulation. The quantitative comparison shows that for the F-Wave solver, the free surface η behaves as the water height in Figure 6.3 did. The wave speeds $|\lambda_y|$ also show good agreement with the wave speeds in Figure 6.3. However, they are shifted downwards for the area where $b = -0.9\text{m}$. This is because η equals 0.0m for $h = 0.9\text{m}$. Thus, the wave speeds in this area reduce to $\sqrt{gh} \approx 2.971\text{m/s}$, instead of 3.132m/s. The Rusanov solver shows similar results as the F-Wave solver except for the artifacts at the bathymetry discontinuity. The Rusanov solver once again undershoots the expected limit for η, as it did for h in the radial dam break scenario. It can be concluded that the non-conservative product works as expected. Also, the F-Wave solver produces qualitatively superior results over bathymetry discontinuities when compared to the Rusanov solver.

6.3. Radial Obstacle Dam Break

A radial dam break with a dry ring around it, which reflects water. The initial conditions are:

$$r = \sqrt{(\frac{N}{2} - x)^2 + (\frac{N}{2} - y)^2}, \tag{6.7}$$

$$h(x, y, 0) = \begin{cases} 1.1\text{m} & \text{if } r \leq \frac{N}{10} \\ 0.0\text{m} & \text{if } 2\frac{N}{10} \leq r \leq 3\frac{N}{10} \\ 1.0\text{m} & \text{otherwise} \end{cases}, \tag{6.8}$$

$$hu(x, y, 0) = hv(x, y, 0) = 0.0\text{m}^2/\text{s}, \tag{6.9}$$

$$b(x, y) = \begin{cases} 2.0\text{m} & \text{if } 2\frac{N}{10} \leq r \leq 3\frac{N}{10} \\ -1.0\text{m} & \text{otherwise} \end{cases}, \tag{6.10}$$

as given in Figure 6.7. As before, x, y are given in meters on a square domain $[0, N] \times [0, N]$.

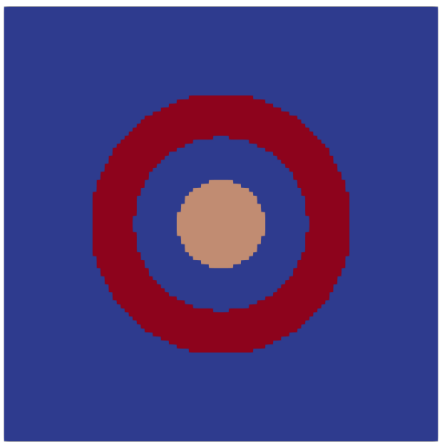


Figure 6.7.: Initial free surface η of the radial obstacle dam break. The logarithmic color scale ranges from blue $= 2.0 \cdot 10^{-4}\text{m}$ to red $= 2.0\text{m}$.

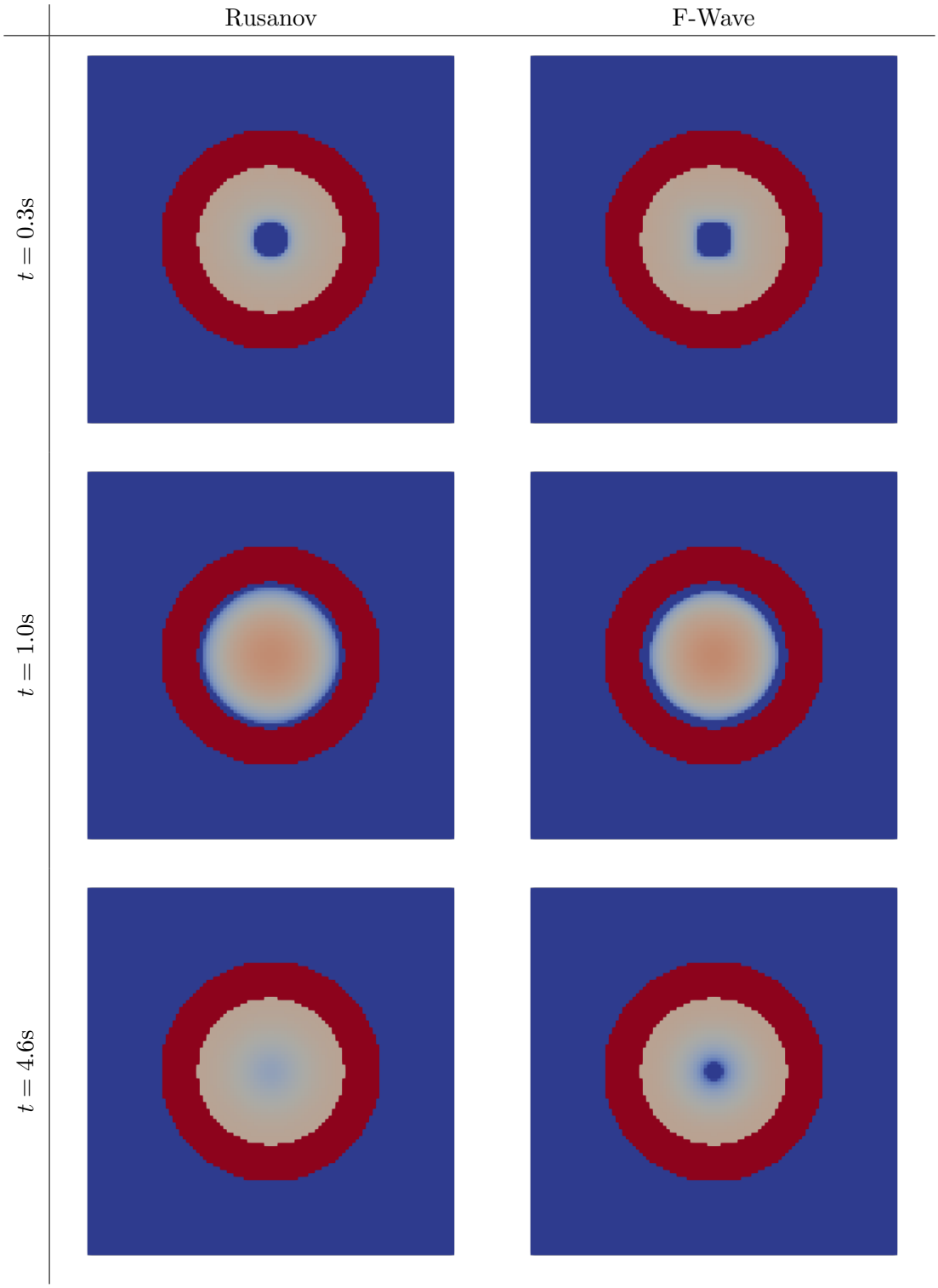


Figure 6.8.: Qualitative comparison of η in the radial obstacle dam break scenario between Rusanov and F-Wave solver. The logarithmic color scale ranges from blue $= 2.0 \cdot 10^{-4}$m to red $= 2.0$m.

Figure 6.9.: Quantitative comparison of η and $|\lambda_y|$ on a logarithmic scale in the radial obstacle dam break scenario between Rusanov and F-Wave solver. The quantities are plotted over the simulated time for $t \in [0.3, 40]$s (continued on the next page).

Water Height
$|\lambda_y|$
$t = 4.6$s
$t = 40$s
eta (Rusanov)
eta (F-Wave)
lambda_y (Rusanov)
lambda_y (F-Wave)

Both solvers are found to properly handle the wet-dry interface on both the qualitative comparison in Figure 6.8, and on the quantitative comparison in Figure 6.9. However, there are some differences. Mainly, the F-Wave solver maintaining the momentum for longer than the Rusanov solver. This can be seen for $t = 4.6$s for example. But both solvers converge on to the same solution, where the water inside the ring is evenly distributed. As this scenario was designed around the wet-dry interface, it can be concluded that both solvers effectively implement the reflective boundary conditions at the wet-dry boundary.

6.4. Artificial Tsunami

This artificial test scenario deals with more complex bathymetry. The initial flux in this scenario is given by an artificial displacement, thus mimicking a real seismogenic tsunami event. The initial conditions (cf. Figure 6.10) are:

$$h(x, y, 0) = 100\text{m}, \tag{6.11}$$

$$hu(x, y, 0) = hv(x, y, 0) = 0.0\text{m}^2/\text{s}, \tag{6.12}$$

$$d(x, y) = 5\text{m} \cdot \sin\left((\frac{x}{500\text{m}} + 1)\pi\right) \cdot (-(\frac{y}{500\text{m}})^2 + 1), \tag{6.13}$$

$$b(x, y) = \begin{cases} -100\text{m} + d(x, y) & \text{if } |x| \leq 500\text{m and } |y| \leq 500\text{m} \\ -100\text{m} & \text{otherwise} \end{cases}. \tag{6.14}$$

$$\tag{6.15}$$

For this scenario, $(x, y) \in [-5000, 5000]\text{m} \times [-5000, 5000]\text{m}$. The scenario is run on a configured volume size of 100m. The problem is solved over a domain $[0, 10000]\text{m} \times [0, 10000]\text{m}$ and all coordinates then shifted down by -5000m. This is done to avoid having to use an offset for the domain.

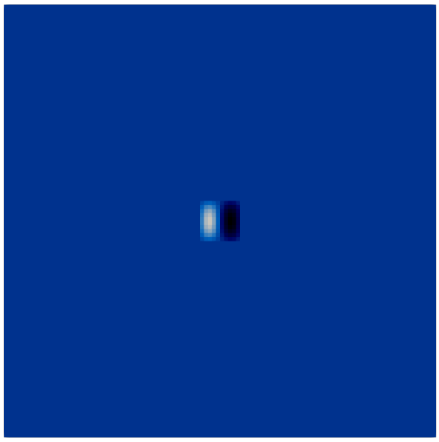


Figure 6.10.: Initial free surface η of the artificial tsunami scenario. Colors range from white = 4.9m to black = -4.9m. Blue corresponds to 0.0m.

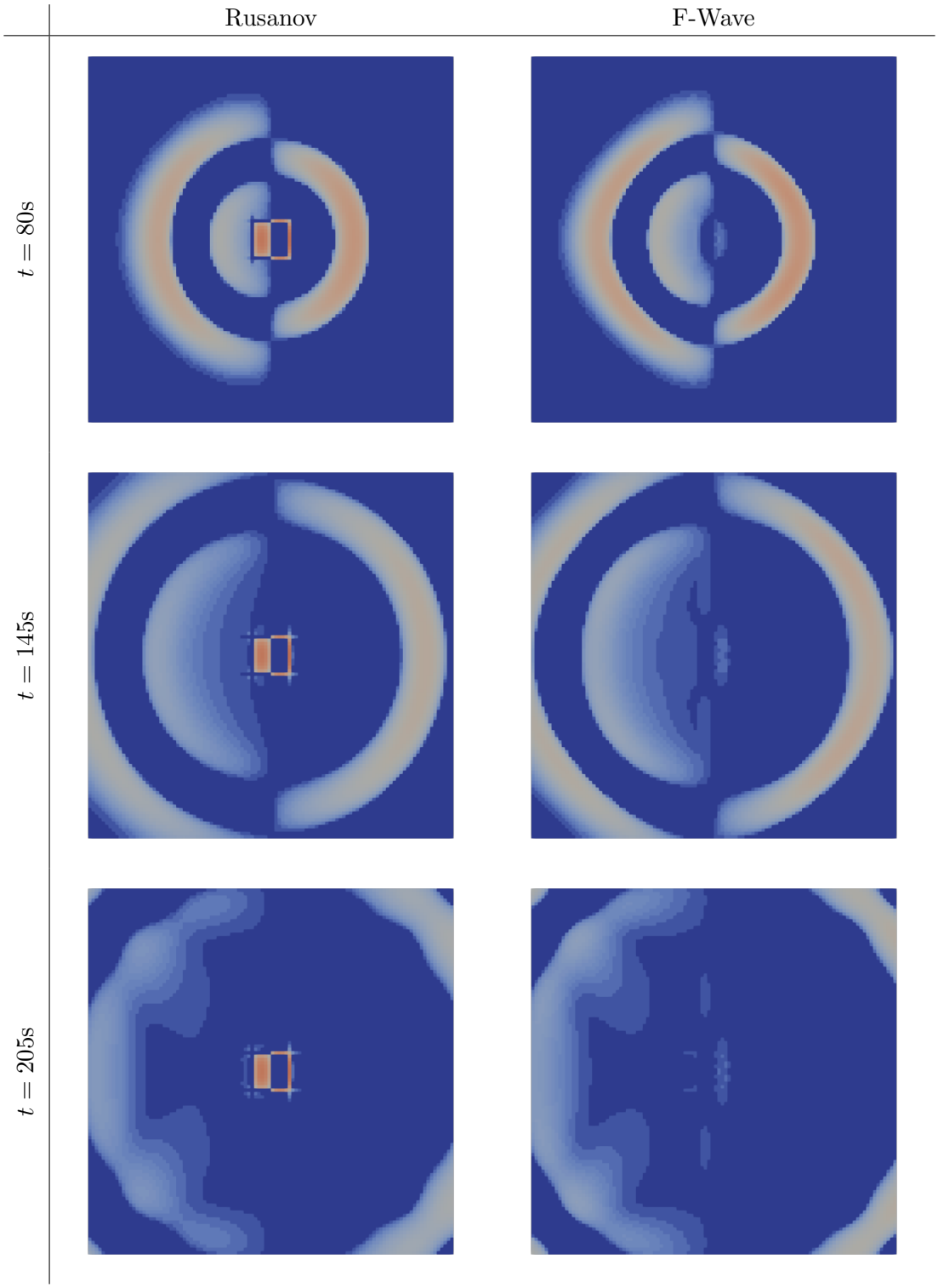


Figure 6.11.: Qualitative comparison of η in the artificial tsunami scenario between Rusanov and F-Wave solver using a logarithmic color scale from red = 4.9m to blue = $4.9 \cdot 10^{-4}$m.

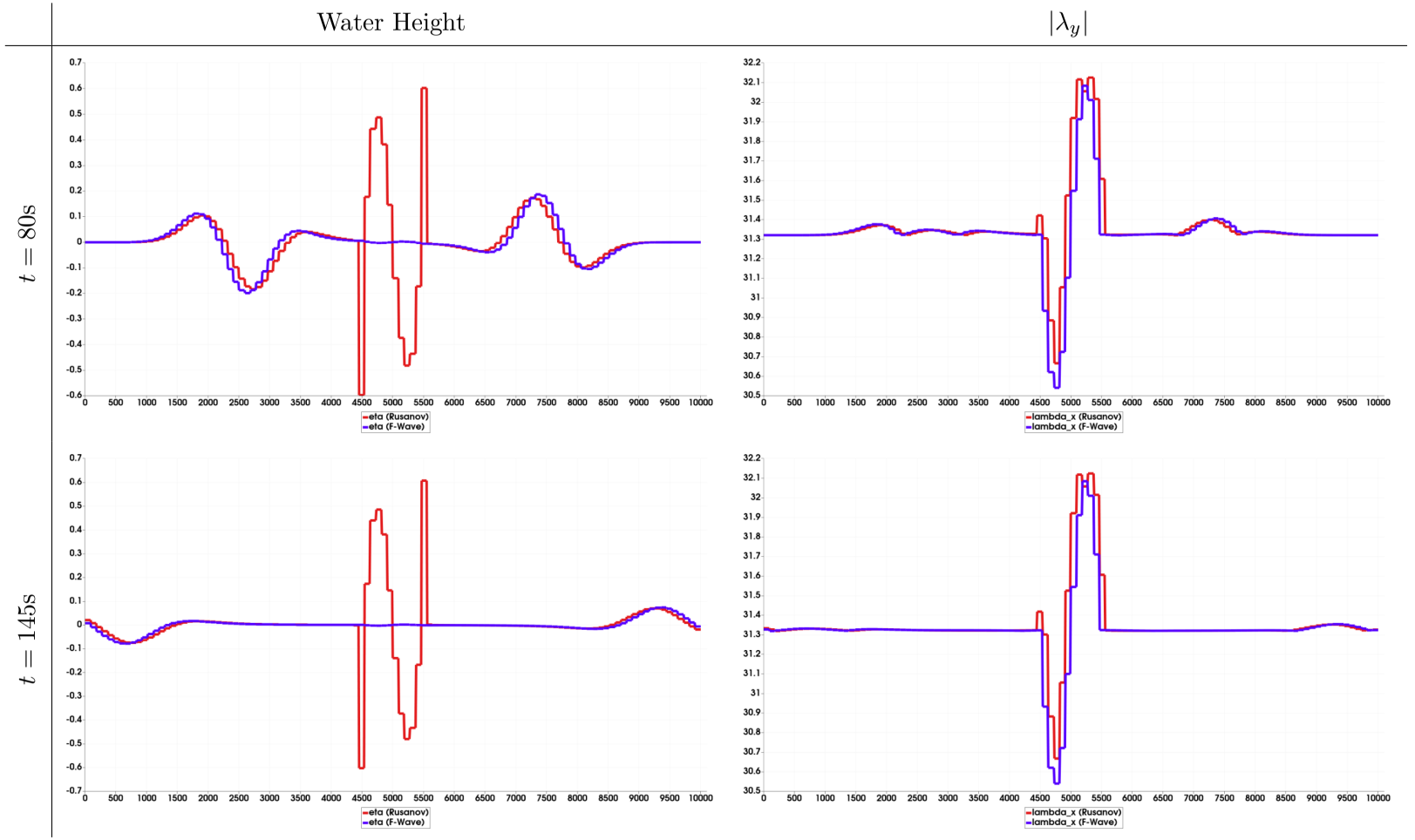


Figure 6.12.: Quantitative comparison of η and $|\lambda_x|$ in the artificial tsunami scenario between Rusanov and F-Wave solver. The values are plotted along a line through the domain from $(0, 5000)$m to $(10000, 5000)$m. The blue line is the F-Wave solver's solution, while the red line is the Rusanov solver's solution. The quantities are plotted over the simulated time for $t \in [80, 600]$s (continued on the next page).

Water Height
$|\lambda_y|$
$t = 205s$
$t = 600s$
lambda_x (Rusanov)
lambda_x (f-Wave)
eta (Rusanov)
eta (f-Wave)

Figure 6.11 shows a similarly pronounced disagreement between the Rusanov and F-Wave solvers over bathymetry discontinuities as the radial bathymetry dam break scenario showed. However, both the Rusanov solver and the F-Wave solver produce artifacts. The F-Wave solver's artifacts are less pronounced than the Rusanov solver's, but they still cause a slight dip in η when looking at the limit in Figure 6.12. This has also been noted for the radial dam break scenario. The wave speeds of the F-Wave solver reach a limit close to $\sqrt{9.81\text{m/s}^2 \cdot 100\text{m}} \approx 31.32\text{m/s}$ throughout the domain, excluding the displaced area. For $x < 0$, the Rusanov solver produced reasonably similar $|\lambda_y|$ to the F-Wave solver. However, the Rusanov solver fails to achieve point symmetric wave speeds over the displacement. As the displacement along the line plot reduces to $5\text{m} \cdot \sin\left(\left(\frac{x}{500\text{m}} + 1\right)\pi\right)$, point symmetry is to be expected. The F-Wave solver is found to produce better approximate solutions over more complex bathymetry than the Rusanov solver.

6.5. Adjoint Solver

The solver for the adjoint problem has first been translated from the FORTRAN code in [25] to Python code. The correctness of this translation has then been verified by plotting the graphs for the one-dimensional continental shelf example.

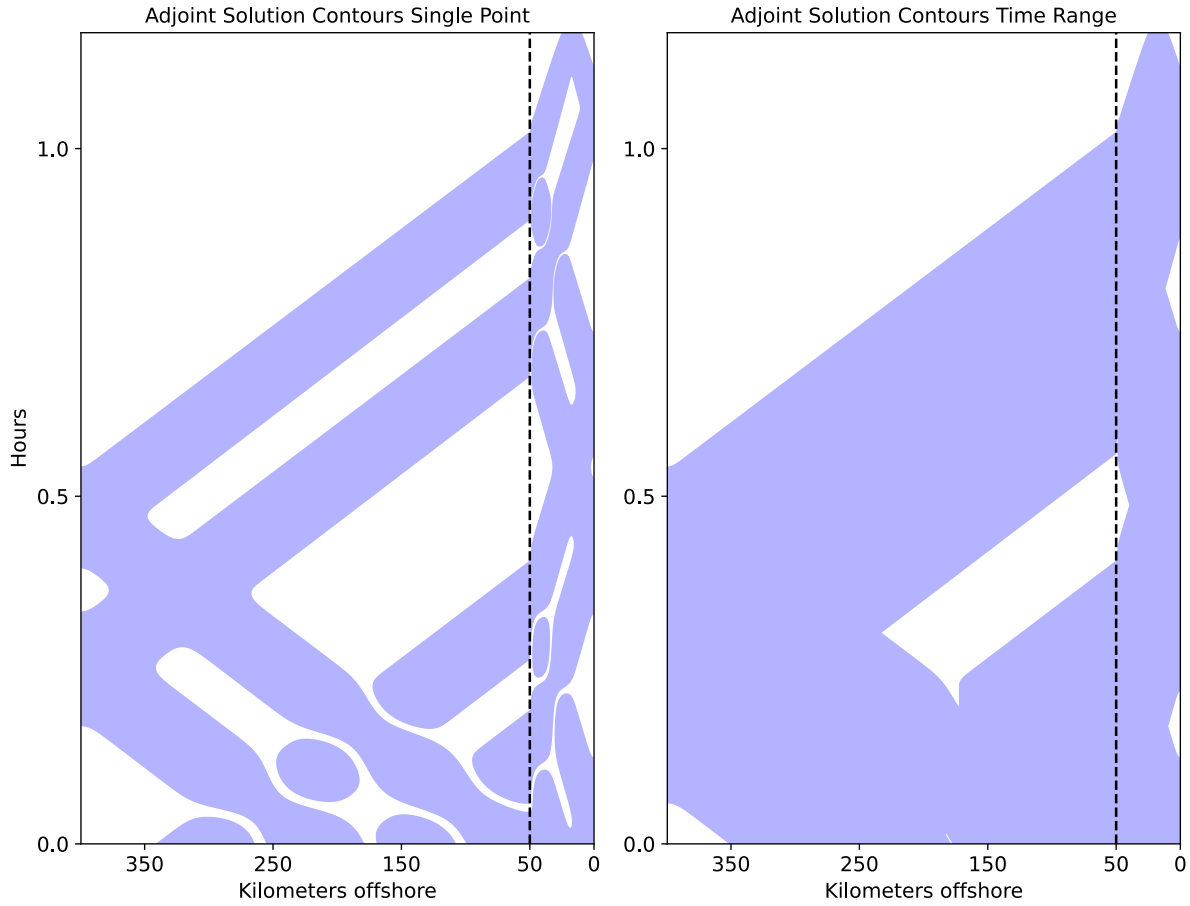


Figure 6.13.: Adjoint solution contours for the one-dimensional continental shelf problem. The problem has been solved until $t = 4200\text{s}$ and shifted in time for $[3800, 4200]\text{s}$.

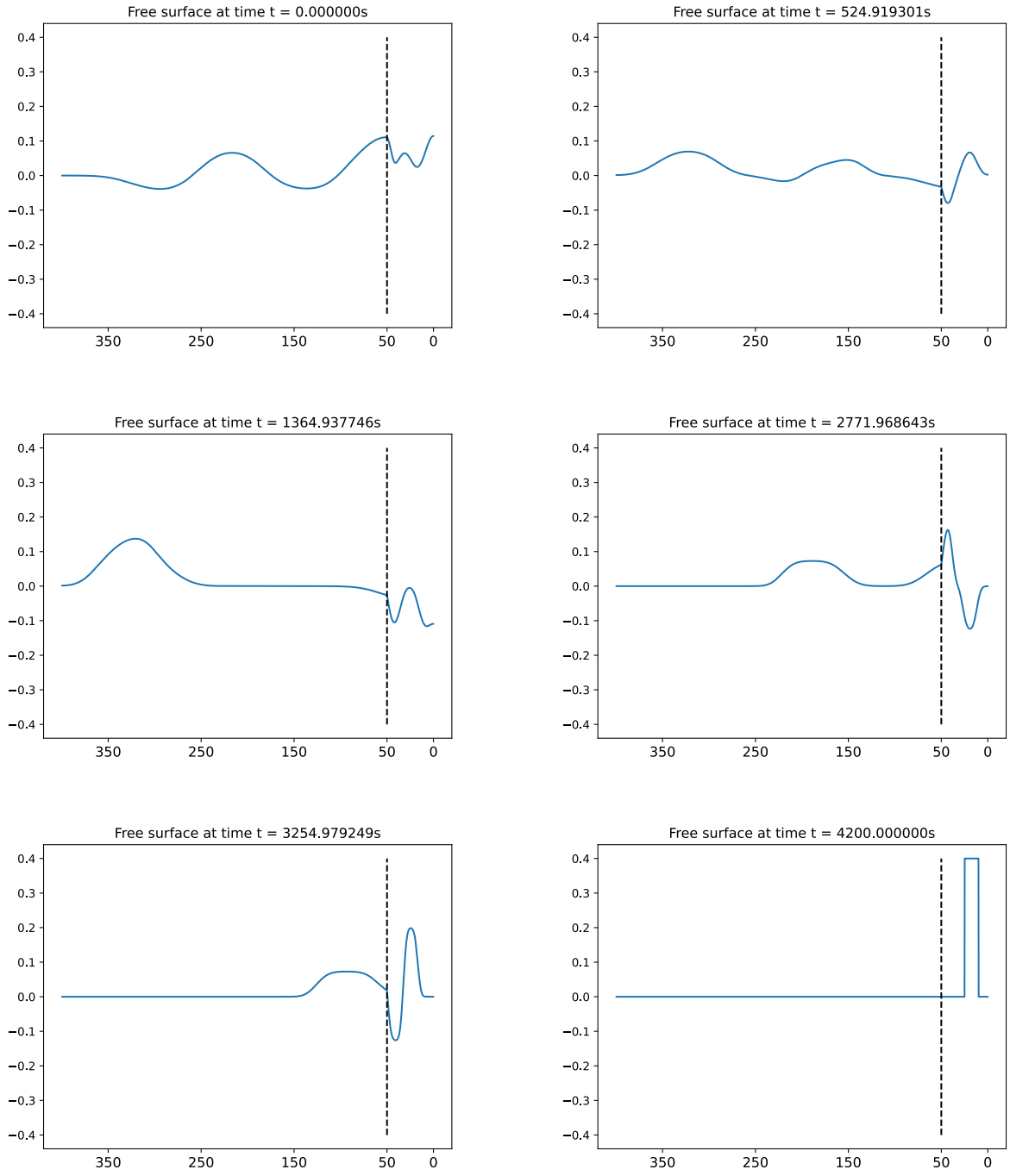


Figure 6.14.: Adjoint solution for the one-dimensional continental shelf problem for $t \in [0, 4200]$s. The dashed line is the discontinuity in the bathymetry. The x-axis is given in kilometers from shore.

The Figures 6.13 and 6.14 show good agreement with the figures given in [25]. The solver used

in ExaHyPE 2 is written in C++ for performance reasons, but virtually a line-by-line equivalent of the solver written in Python. The same is also true for the dimensional splitting scheme (cf. Algorithm 7), which is used in its one-dimensional form in the Python code.

6.6. Artificial Tsunami with Adaptive Mesh Refinement

Using the artificial tsunami scenario from Section 6.4, the two AMR algorithms from Section 4 can be verified. The two algorithms are tested with a refinement tolerance of 10^{-3}m (guided AMR, cf. Algorithms 5, 8) and 10^{-2}m (surface-flagging, cf. Algorithm 4). The configured maximum volume size for both AMR approaches is 1000m (cf. Figure 6.15). The adjoint simulation is run on a grid with a maximum volume size of 100m and without AMR. A circular AoI at $(-4900, 0)$m with a radius of 100m was chosen. The inner product of the guided AMR is calculated for a time range size of 430s. The two tests are run with 4 levels of AMR using the F-Wave solver.

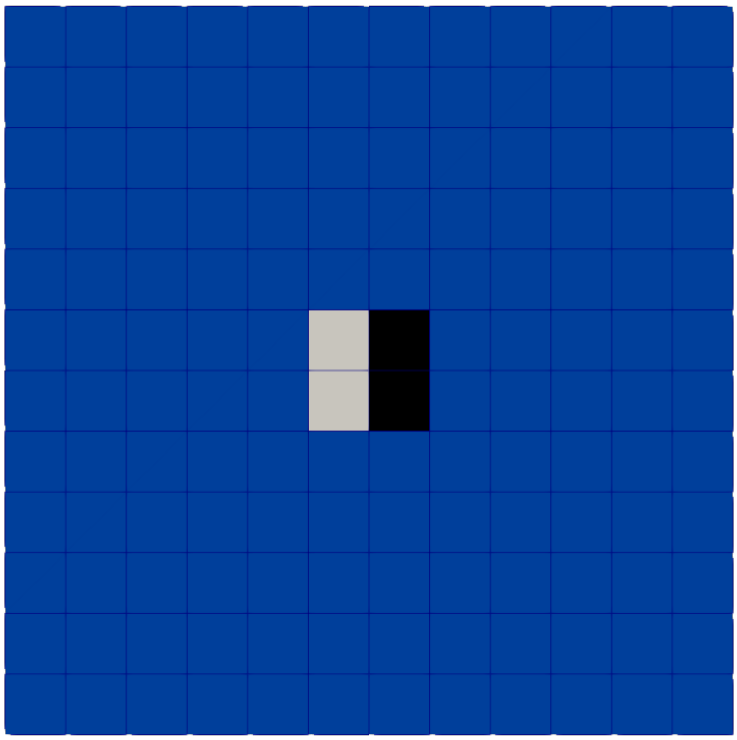
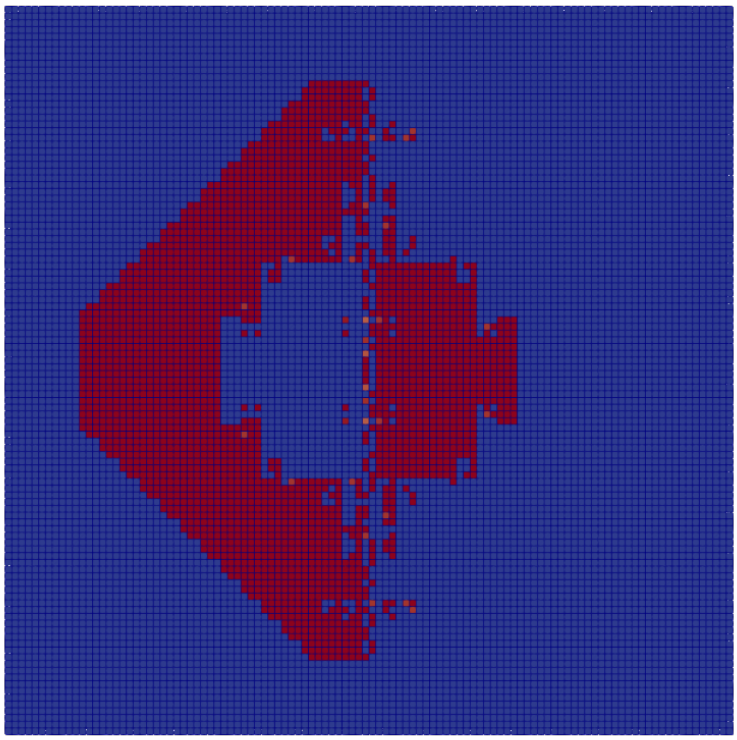
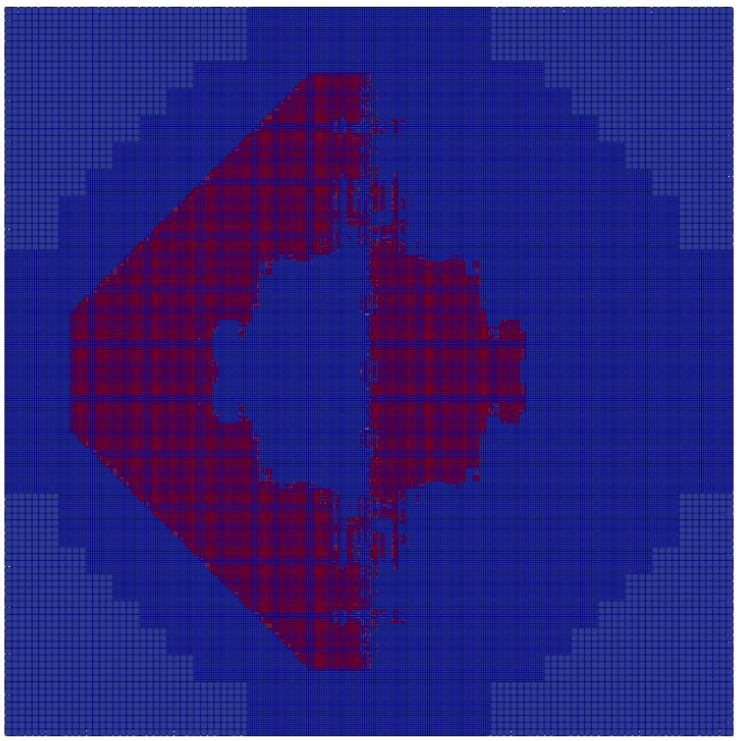


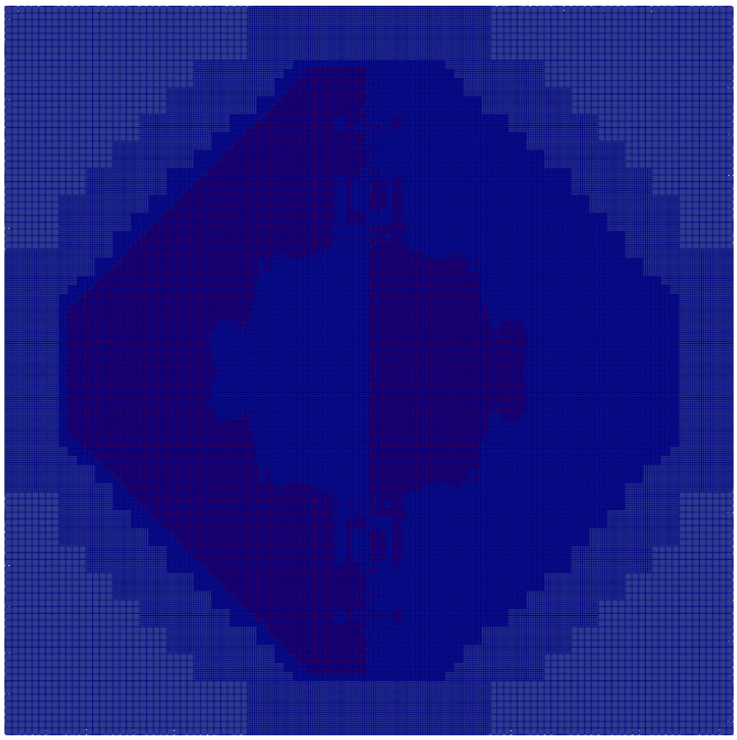
Figure 6.15.: Initial conditions for AMR with the artificial tsunami scenario.



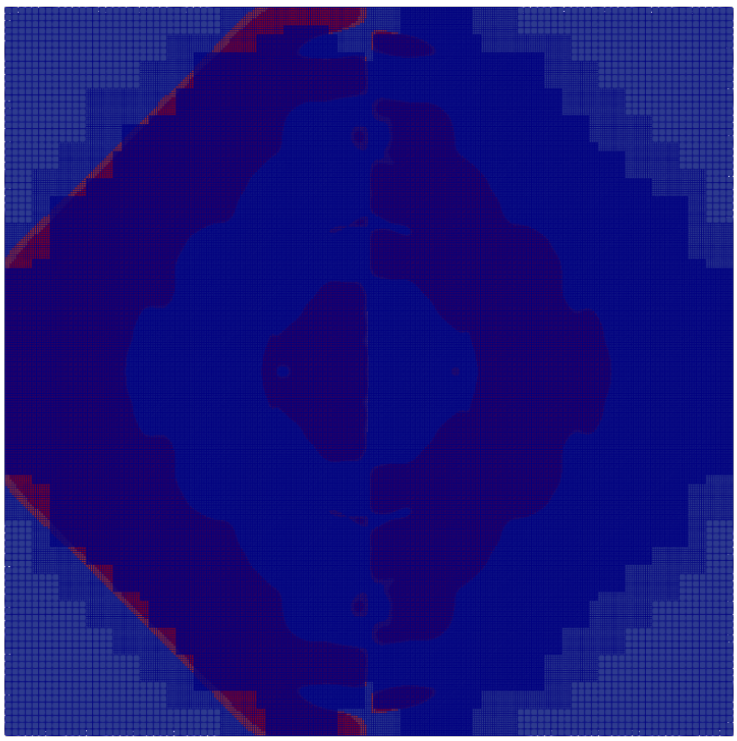
(a) Surface-Flagging at $t = 46$s.



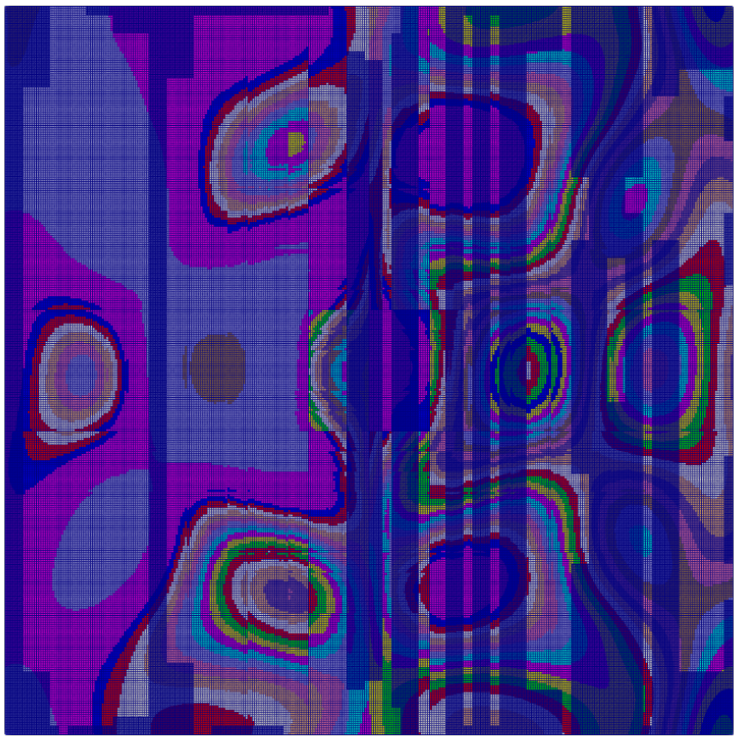
(b) Surface-Flagging at $t = 50$s.



(c) Surface-Flagging at $t = 52$s.



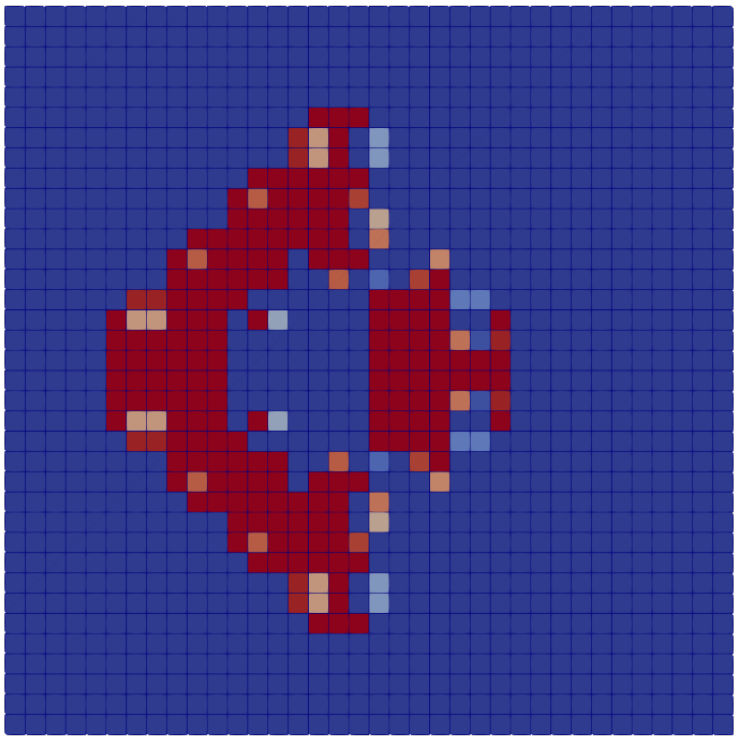
(d) Surface-Flagging at $t = 85.5$s.



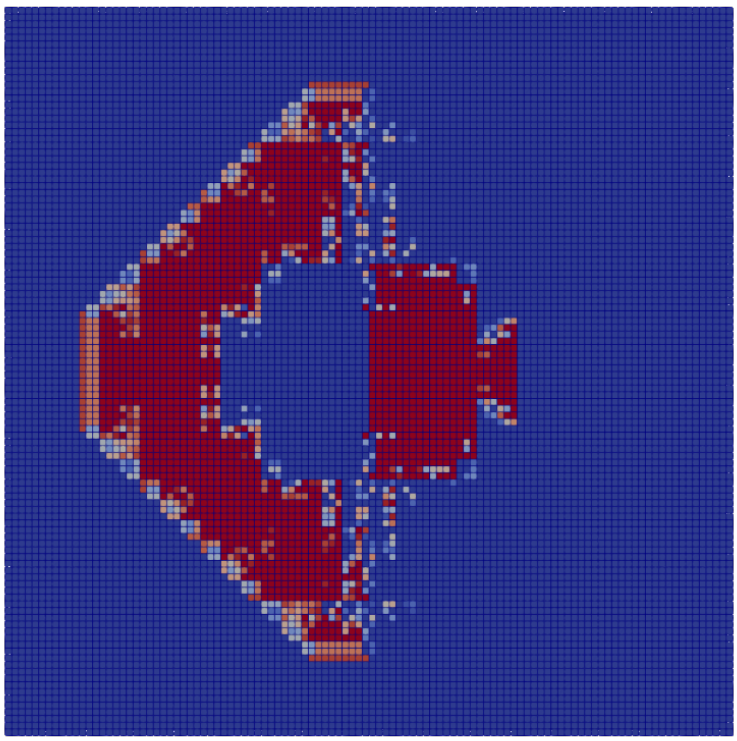
(e) Surface-Flagging at $t = 515$s. Colors represent distinct value categories.

Figure 6.16.: Artificial tsunami using the surface-flagging approach. If not stated otherwise, the figures depict the water height h in the artificial tsunami scenario on a linear color scale ranging from red = 100.016m to blue = 100m.

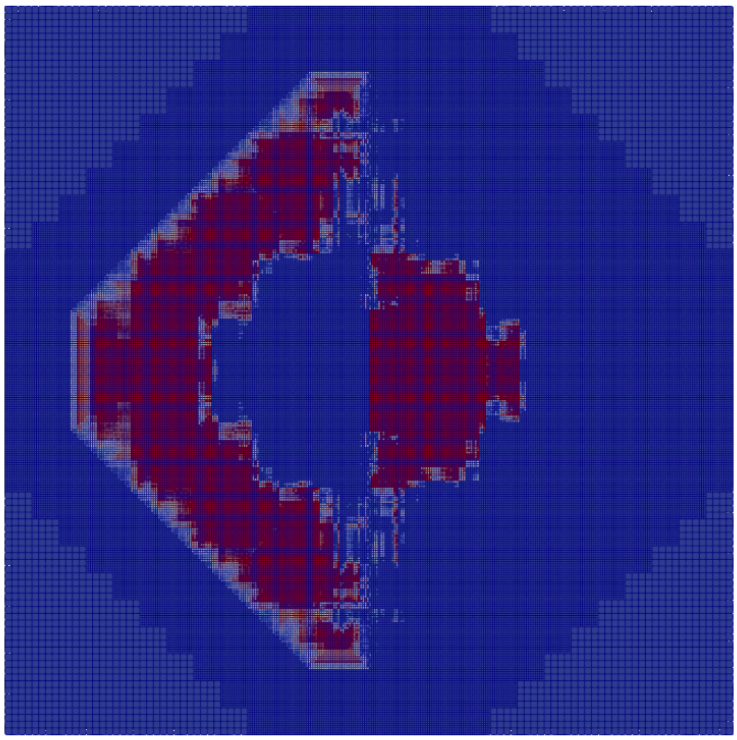
Figure 6.16 depicts that the AMR does refine volumes. However, we can hardly make any statements about which volume is refined. At this point, we assume a flawed AMR in ExaHyPE 2. In general, all volumes are refined by two levels at first, as can be seen for $t = 46$s. It seemingly does not matter, whether the wave has actually reached these volumes or not. For $t = 50$s, more refinement can be seen in volumes that have not seen a change in η. For $t = 52$s, the refinement has reached the finest level, now somewhat matching the wave front on the left side of the domain, but still greatly overrefining the right side of the domain. During the simulation, the wave front moves faster than the AMR, for example at $t = 85.5$s. This is due to Peano needing at least 3 sweeps over the grid to realize refinement for FV solvers. Thus, the wave outpaces the refinement. For greater t, the AMR does not converge towards any coherent structure. As can be seen for $t = 515$s, the AMR keeps strips of different values refined, while coarsening neighboring cells with the same values.



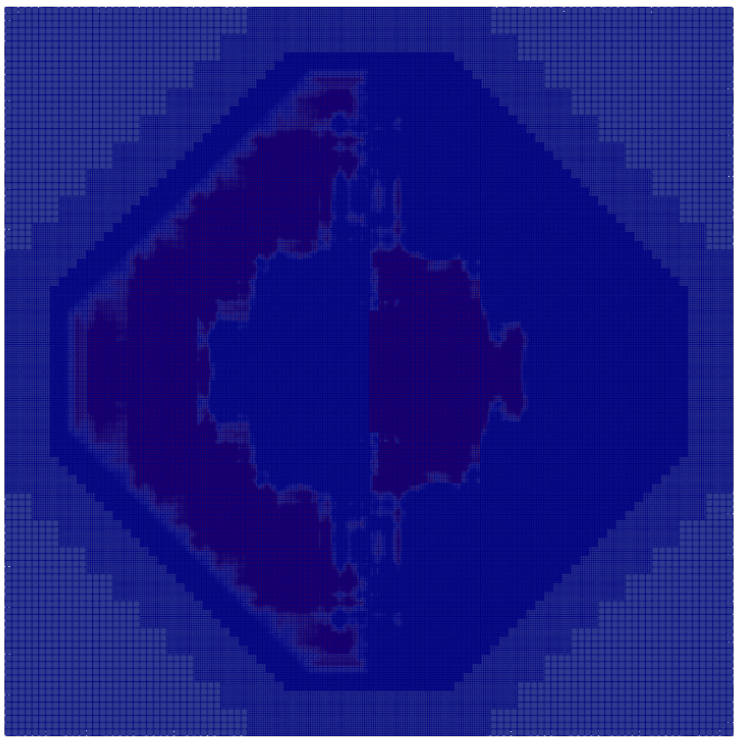
(a) Guided AMR at $t = 32$s.



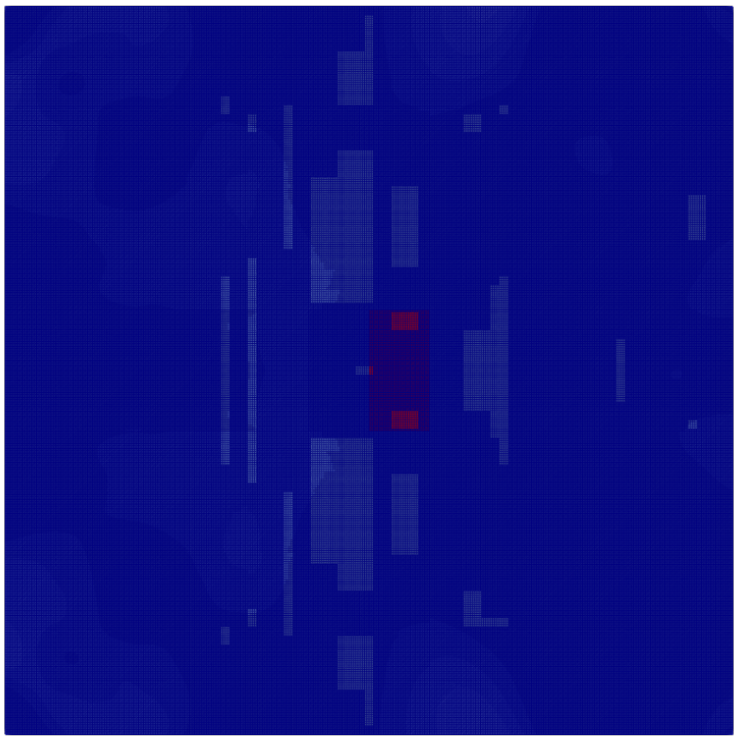
(b) Guided AMR at $t = 46$s.



(c) Guided AMR at $t = 50$s.



(d) Guided AMR at $t = 52.7$s.



(e) Guided AMR at $t = 327$s.

Figure 6.17.: Artificial tsunami using the guided AMR approach. The figures depict the water height h in the artificial tsunami scenario on a linear color scale ranging from red = 100.016m to blue = 100m.

The guided AMR in Figure 6.17 looks similar to the surface-flagging approach of Figure 6.16. The only noticeable difference arising is that the wave front in the guided AMR does not outpace the refinement. For $t = 327$s, the guided AMR has not reached a state resembling any expected AMR outcome. As the surface-flagging also failed to achieve this after even more time, no further conclusions can be made about the guided AMR approach implemented in ExaHyPE 2. But with the results from Section 6.5, there is good reason to believe that the implementation works as intended, provided that the AMR also would be working correctly.

6.7. UM-Bridge Model Evaluation Using the Artificial Tsunami

The UM-Bridge model server provided as part of this work, which has been discussed in Chapter 5, is tested and verified by requesting different evaluations of the artificial tsunami scenario.

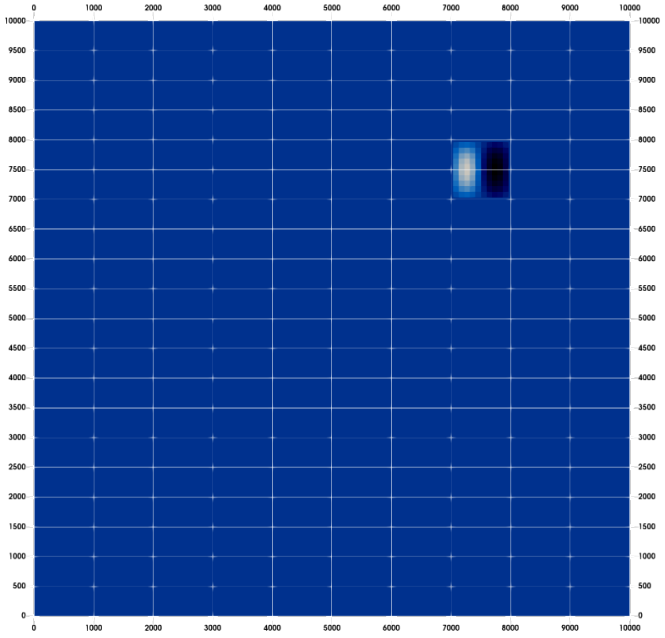


Figure 6.18.: Displacement of the artificial tsunami scenario, with the origin moved from $(0, 0)$m to $(2500, 2500)$m.

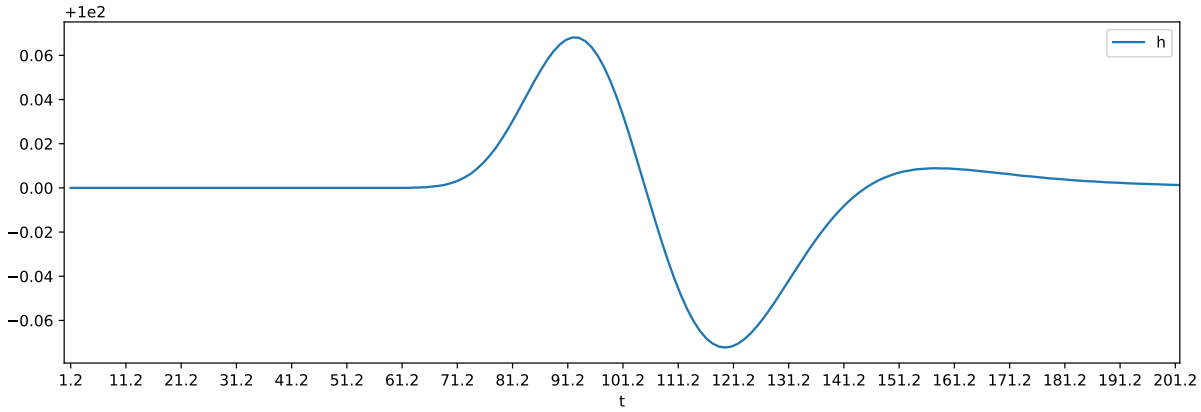


Figure 6.19.: Water height of the artificial tsunami with origin moved to $(2500, 2500)$m for $t \in [0, 202]$s. Values taken from probe at $(5000, 5000)$m, y-axis showing the difference to 100m.

For the scenario depicted in Figure 6.18, the server reported a maximum water height of 100.0681m for $t = 92.00676$s for a probe in the center of the domain at $(5000, 5000)$m. This measurement can be confirmed when comparing with Figure 6.19.

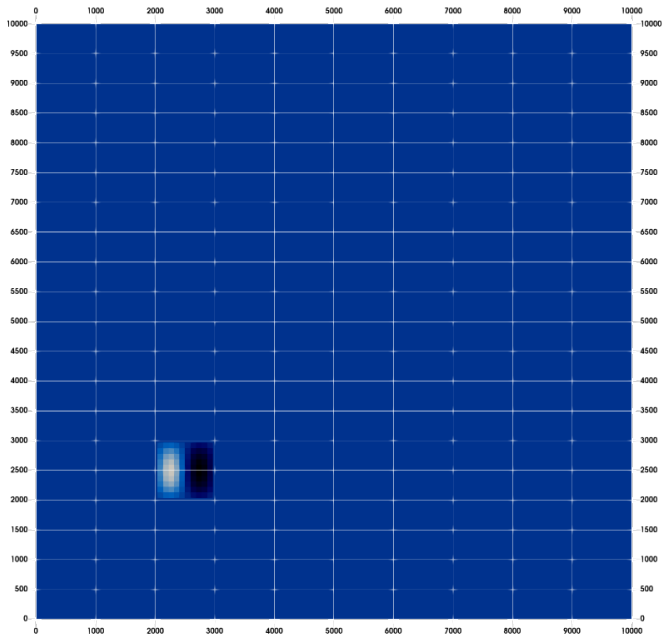


Figure 6.20.: Displacement of the artificial tsunami scenario, with the origin moved from $(0, 0)$m to $(-2500, -2500)$m.

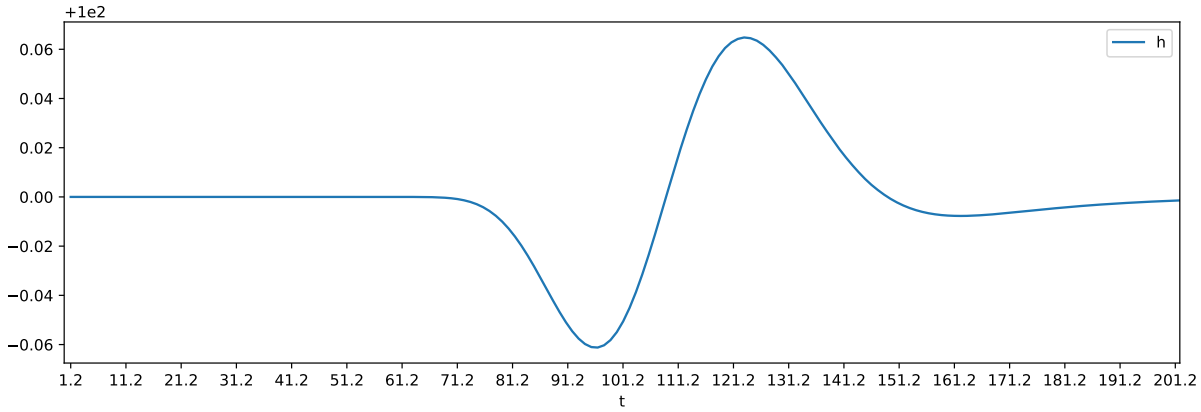


Figure 6.21.: Water height of the artificial tsunami with origin moved to $(-2500, -2500)$m for $t \in [0, 202]$s. Values taken from probe at $(5000, 5000)$m, y-axis showing the difference to 100m.

When moving the displacement to $(-2500, -2500)$m (cf. Figure 6.20), the model server reports a maximum water height for the probe at $(5000, 5000)$m of 100.0648m for $t = 123.1139$s. This measurement is in agreement with the complete data recorded by the probe (cf. Figure 6.21). It can thus be concluded that the displacement can be correctly moved within the domain and that the model server functions as expected.

6.8. Tōhoku Tsunami

To test the F-Wave solver in a real world application, the 2011 Tōhoku tsunami was chosen. The solver is tested on a grid of size $(7 \cdot 10^6, 4 \cdot 10^6)$m and a configured volume size of 10^4m. No AMR was used. The initial conditions resulting from the earthquake can be seen in Figure 6.22.

Figure 6.22.: Initial conditions of the Tōhoku tsunami. Displaying the free surface η on a logarithmic scale.

When approximating the wave speed for the deepest bathymetry of -9843.52m and a flat surface, a maximum wave speed of $\sqrt{gh} \approx 310.75$m/s is reached. As tsunami reach an estimated maximum speed of 800km/h $= 222.2$m/s, the waves within the simulation should reach the shore faster than in reality. Taking the city of Sōma in Fukushima Prefecture as an example, the real tsunami reached it within 9min coming from its origin about 134km away[2]. However, in the simulation, Sōma is only reached after about 36min (cf. Figure 6.23). This does neither match the fastest approximation using 310.75m/s, which would imply a travel time of 429s, nor the real time of about 9min. Instead, the results lead to an average wave speed of only 62.04m/s for the wave front moving towards the shore.

[2] `https://www.ngdc.noaa.gov/hazel/view/hazards/tsunami/runup-more-info/19241`, accessed August 22nd, 2023

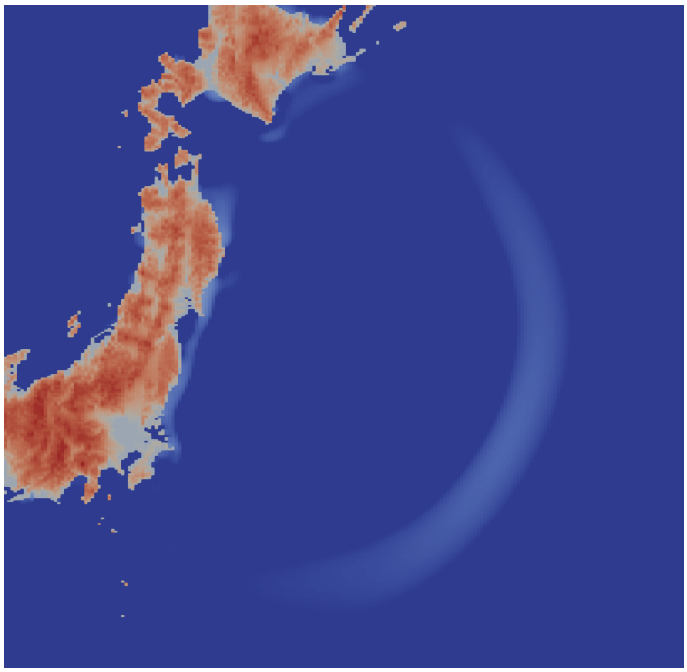


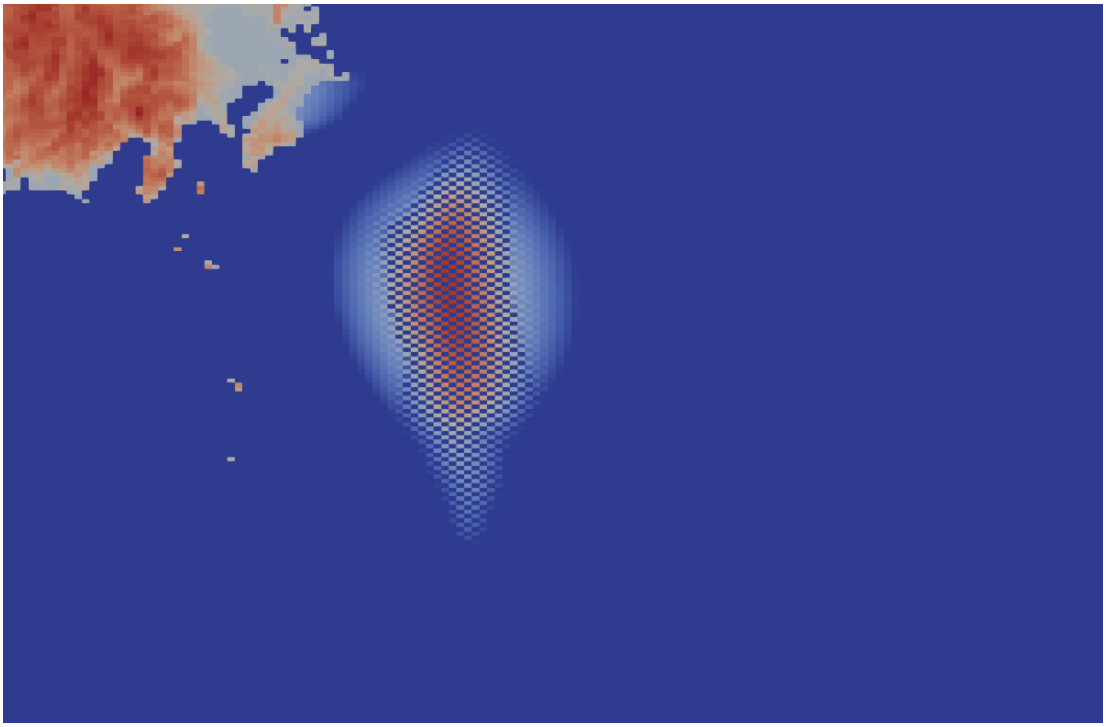
Figure 6.23.: Tōhoku tsunami approaching Sōma at $t = 2160$s with a TSR factor of 0.4.

The simulation qualitatively looks promising until artifacts start forming from around $t = 2940$s, as can be seen in Figure 6.25. These artifacts move in space and time when changing the TSR factor (cf. Figures 6.24, 6.26). When the factor is raised to 0.5, the artifacts appear much earlier and develop much more violently, as can be seen in when comparing Figures 6.25 and 6.26. For a TSR factor of 0.3, no artifacts can be observed for a maximum simulation time of 36000s. However, it can not be ruled out that artifacts appear later on in the simulation. In general, the simulation does not match the actual event, regardless of the chosen TSR factor.

TSR Factor	0.3	0.4	0.5	0.6	0.7	0.8	0.9
artifacts start appearing at	? > 36000s	3000s	186s	83s	65s	56s	42s

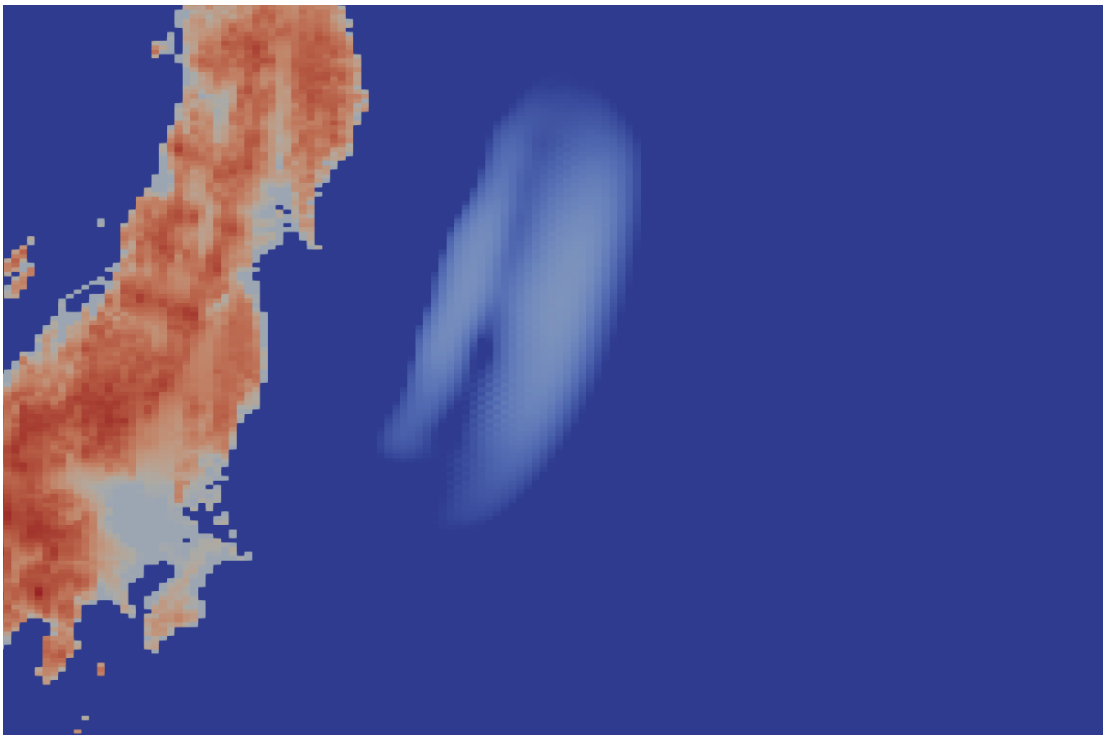
Figure 6.24.: Influence of varying TSR factors on artifacts of the Tōhoku tsunami.

(a) Artifacts for $t = 3000$s with a TSR factor of 0.4.

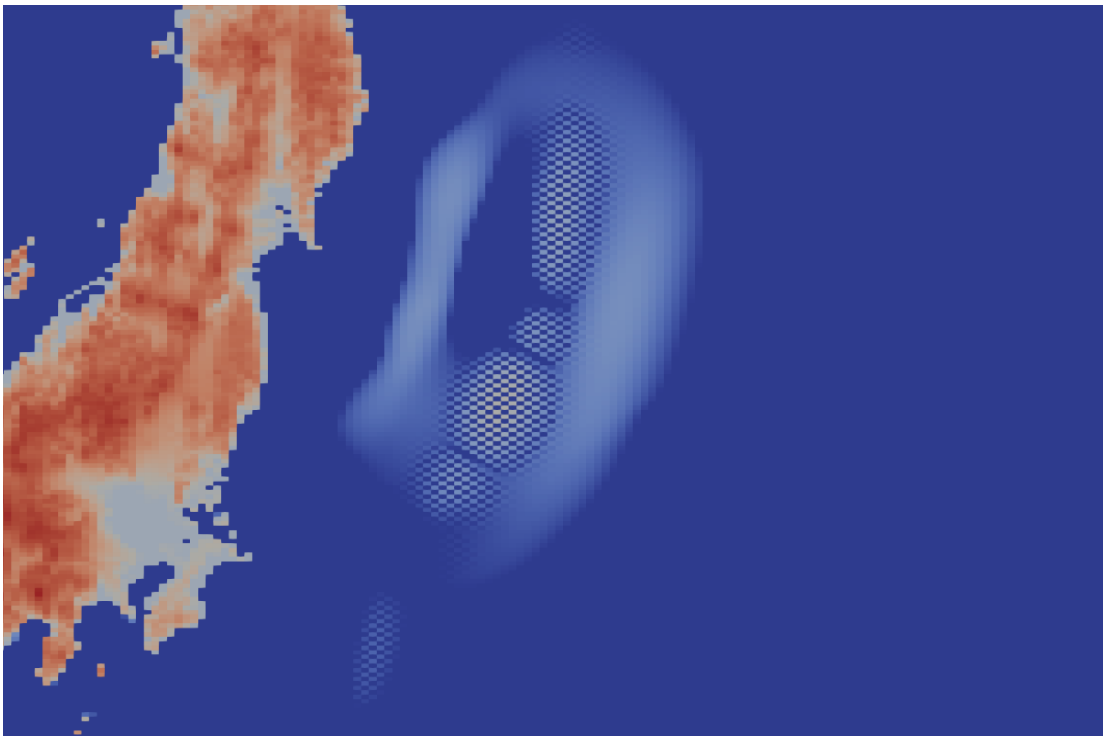


(b) Artifacts for $t = 4200$s with a TSR factor of 0.4.

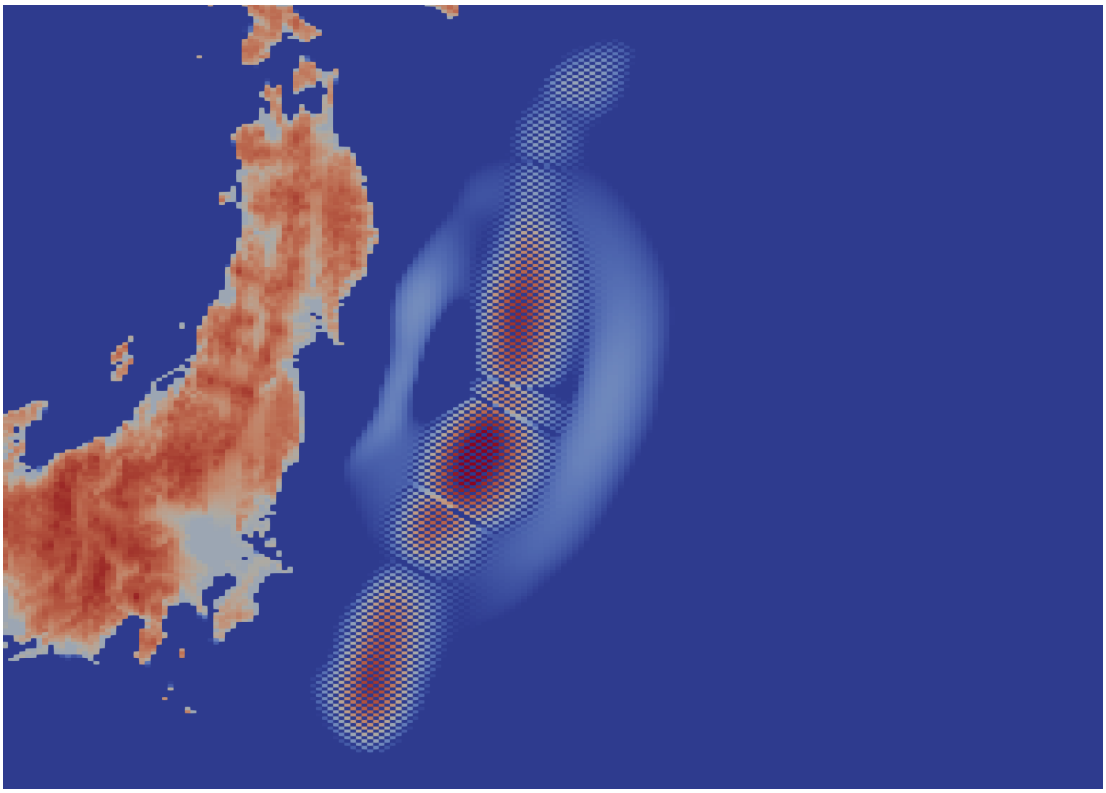
Figure 6.25.: Artifacts of the Tōhoku tsunami south-east of Chiba Prefecture with a TSR factor of 0.4. The figure shows the free surface η on a logarithmic scale.



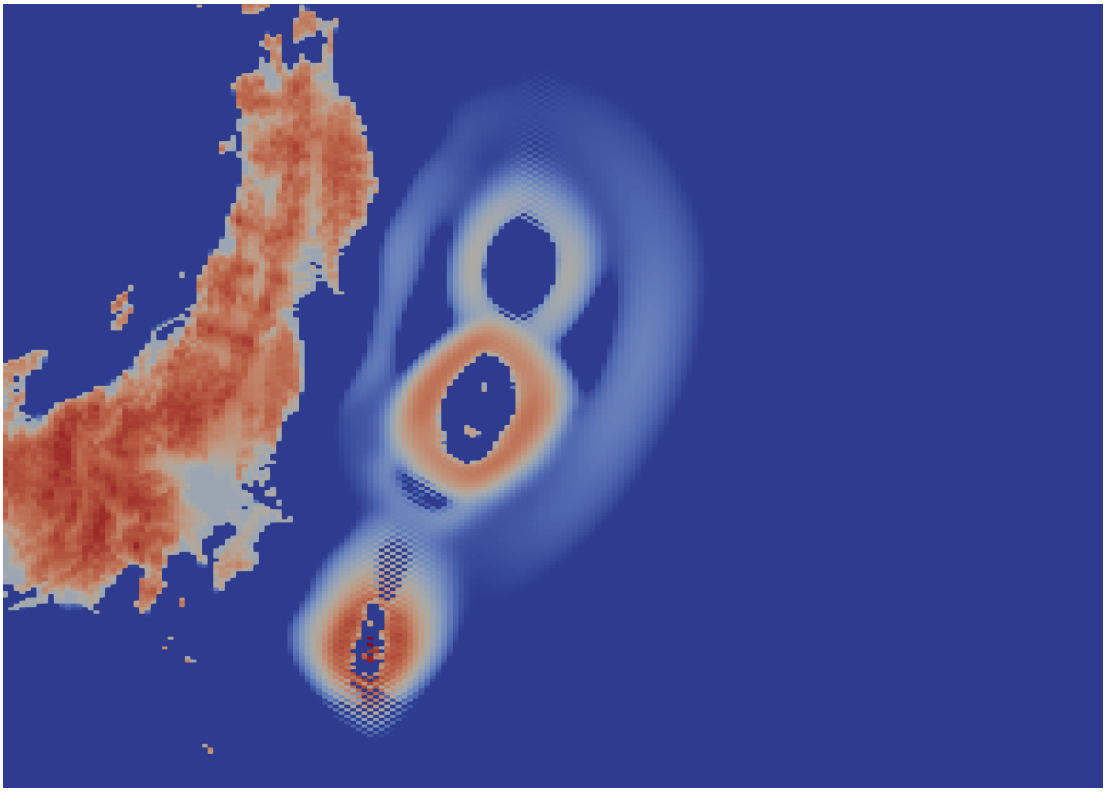
(a) Artifacts for $t = 186$s with a TSR factor of 0.5.



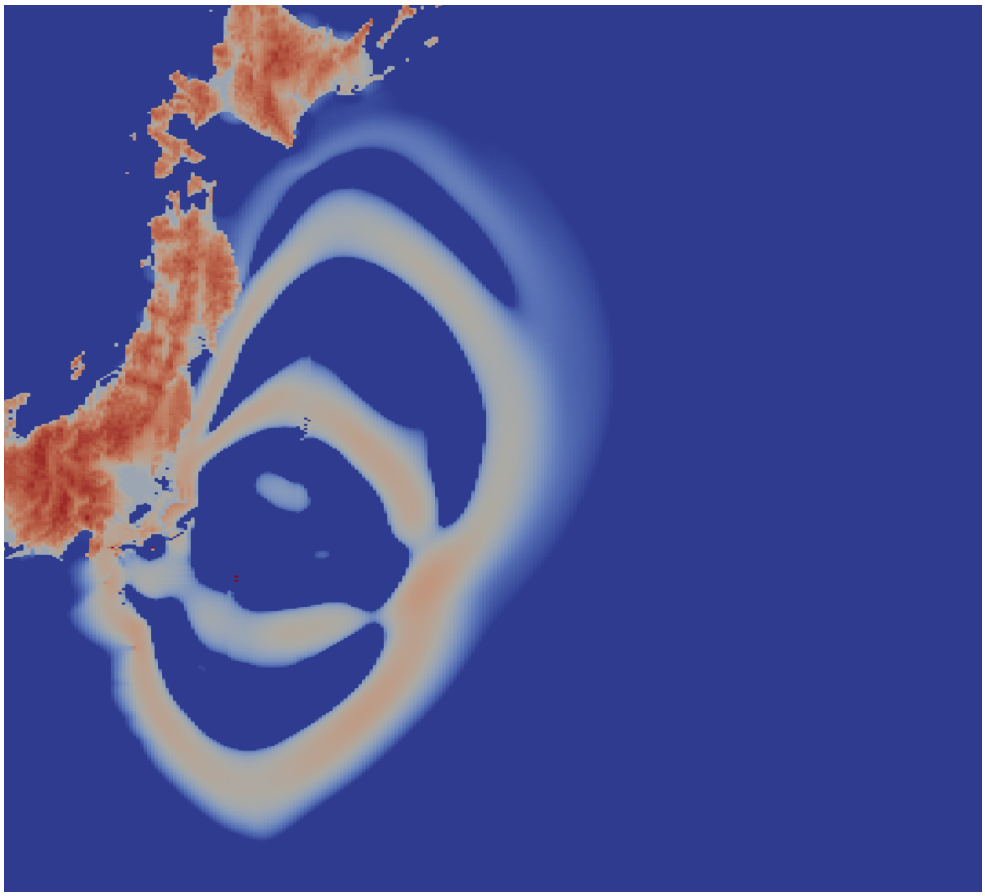
(b) Artifacts for $t = 407$s with a TSR factor of 0.5.



(c) Artifacts for $t = 640$s with a TSR factor of 0.5.



(d) Artifacts for $t = 822$s with a TSR factor of 0.5.



(e) Artifacts for $t = 2162$s with a TSR factor of 0.5.

Figure 6.26.: Artifacts of the Tōhoku tsunami for a TSR factor of 0.5. The figure shows the free surface η on a logarithmic scale.

As the wave speeds in previous scenarios were found to match the expected speeds, this bug is also expected to have its root course outside the SWE application. This bug can also occur in artificial test scenarios like the artificial tsunami scenario when choosing higher TSR factors than 0.4, independent of the chosen solver, which most notably also includes ExaHyPE 2's Rusanov solver.

7. Conclusion and Outlook

This work set out to provide an UQ workflow within the re-written ExaHyPE 2 engine using the SWE. An SWE application has been implemented in ExaHyPE 2. We verified the implementation using multiple different artificial scenarios, which test individual characteristics of the SWE and their implementation. As the predefined Rusanov solver within ExaHyPE 2 is not sufficient for real world applications, user-defined solvers have been re-introduced to ExaHyPE 2. The user-defined F-Wave solver implementation has been verified together with the SWE application. Additionally, the guided AMR approach of Davis and LeVeque has also been implemented in the SWE application. The algorithms have been verified outside ExaHyPE 2. Within ExaHyPE 2, it was shown that the AMR of ExaHyPE 2 is still erroneous. ExaHyPE 2 tends to overrefine most of the domain in the tested scenario. Additionally, no meaningful coarsening of volumes could be observed. This was also confirmed for the much simpler surface-flagging, proving that the fault does not lie with the SWE application. Finally, based on the SWE application, an UM-Bridge model server has been implemented. Using an artificial test scenario, its functionality has been confirmed. As UQ for the SWE typically includes tsunami simulations, such capabilities were tested as well. It was shown that while matching expected limits for various quantities in the artificial test scenarios, real world applications such as the Tōhoku tsunami could not be simulated. In our experiments, we observed the formation of artifacts, which move in time and space depending on the relaxation factor used for the CFL condition. This points toward a more general problem within the ExaHyPE 2 engine, which is most definitely deserving of further attention. Future work should therefore be focused on resolving the issues around the AMR and tsunami simulation. Based on these improvements, real UQ can be conducted using ExaHyPE 2 and the guided AMR. As the Runge-Kutta Discontinuous Galerkin (RKDG) and Arbitrary Derivative Discontinuous Galerkin (ADER-DG) solvers are implemented in ExaHyPE 2, these schemes could be put together with the guided AMR approach and be coupled with an UQ code for even larger, high-resolution UQ.

List of Figures

List of Algorithms

Code Listings

Bibliography

[1] Shuji Seto and Fumihiko Imamura. Classification of tsunami deaths by modifying ICD-10 categories in the 2011 Tohoku earthquake tsunami - A case study in Miyagi prefecture. *International Journal of Disaster Risk Reduction*, 50:101743, 2020.

[2] Thorne Lay, Hiroo Kanamori, Charles J Ammon, Meredith Nettles, Steven N Ward, Richard C Aster, Susan L Beck, Susan L Bilek, Michael R Brudzinski, Rhett Butler, et al. The great Sumatra-Andaman earthquake of 26 december 2004. *science*, 308(5725):1127–1133, 2005.

[3] Alexander Rudloff, Jörn Lauterjung, Ute Münch, and S Tinti. Preface" The GITEWS Project (German-Indonesian Tsunami Early Warning System)". *Natural Hazards and Earth System Sciences*, 9(4):1381–1382, 2009.

[4] Arnau Folch, Claudia Abril, Michael Afanasiev, Giorgio Amati, Michael Bader, Rosa M. Badia, Hafize B. Bayraktar, Sara Barsotti, Roberto Basili, Fabrizio Bernardi, Christian Boehm, Beatriz Brizuela, Federico Brogi, Eduardo Cabrera, Emanuele Casarotti, Manuel J. Castro, Matteo Cerminara, Antonella Cirella, Alexey Cheptsov, Javier Conejero, Antonio Costa, Marc de la Asunción, Josep de la Puente, Marco Djuric, Ravil Dorozhinskii, Gabriela Espinosa, Tomaso Esposti-Ongaro, Joan Farnós, Nathalie Favretto-Cristini, Andreas Fichtner, Alexandre Fournier, Alice-Agnes Gabriel, Jean-Matthieu Gallard, Steven J. Gibbons, Sylfest Glimsdal, José Manuel González-Vida, Jose Gracia, Rose Gregorio, Natalia Gutierrez, Benedikt Halldorsson, Okba Hamitou, Guillaume Houzeaux, Stephan Jaure, Mouloud Kessar, Lukas Krenz, Lion Krischer, Soline Laforet, Piero Lanucara, Bo Li, Maria Concetta Lorenzino, Stefano Lorito, Finn Løvholt, Giovanni Macedonio, Jorge Macías, Guillermo Marín, Beatriz Martínez Montesinos, Leonardo Mingari, Geneviève Moguilny, Vadim Montellier, Marisol Monterrubio-Velasco, Georges Emmanuel Moulard, Masaru Nagaso, Massimo Nazaria, Christoph Niethammer, Federica Pardini, Marta Pienkowska, Luca Pizzimenti, Natalia Poiata, Leonhard Rannabauer, Otilio Rojas, Juan Esteban Rodriguez, Fabrizio Romano, Oleksandr Rudyy, Vittorio Ruggiero, Philipp Samfass, Carlos Sánchez-Linares, Sabrina Sanchez, Laura Sandri, Antonio Scala, Nathanael Schaeffer, Joseph Schuchart, Jacopo Selva, Amadine Sergeant, Angela Stallone, Matteo Taroni, Solvi Thrastarson, Manuel Titos, Nadia Tonelllo, Roberto Tonini, Thomas Ulrich, Jean-Pierre Vilotte, Malte Vöge, Manuela Volpe, Sara Aniko Wirp, and Uwe Wössner. The EU Center of Excellence for Exascale in Solid Earth (ChEESE): Implementation, results, and roadmap for the second phase. *Future Generation Computer Systems*, 146:47–61, 2023.

[5] Anne Reinarz, Dominic E. Charrier, Michael Bader, Luke Bovard, Michael Dumbser, Kenneth Duru, Francesco Fambri, Alice-Agnes Gabriel, Jean-Matthieu Gallard, Sven Köppel, Lukas Krenz, Leonhard Rannabauer, Luciano Rezzolla, Philipp Samfass, Maurizio Tavelli, and Tobias Weinzierl. ExaHyPE: An engine for parallel dynamically adaptive simulations of wave problems. *Computer Physics Communications*, 254:107251, 2020.

[6] Linus Seelinger, Vivian Cheng-Seelinger, Andrew Davis, Matthew Parno, and Anne Reinarz. UM-Bridge: Uncertainty quantification and modeling bridge. *Journal of Open Source Software*, 8(83):4748, 2023.

[7] Holger Schulz, Gonzalo Brito Gadeschi, Oleksandr Rudyy, and Tobias Weinzierl. Task Inefficiency Patterns for a Wave Equation Solver. In *OpenMP: Enabling Massive Node-Level Parallelism*, pages 111–124. Springer International Publishing, 2021.

[8] Mario Wille, Tobias Weinzierl, Gonzalo Brito Gadeschi, and Michael Bader. Efficient GPU Offloading with OpenMP for a Hyperbolic Finite Volume Solver on Dynamically Adaptive Meshes. In Abhinav Bhatele, Jeff Hammond, Marc Baboulin, and Carola Kruse, editors, *High Performance Computing*, pages 65–85, Cham, 2023. Springer Nature Switzerland.

[9] Chung Ming Loi and Tobias Weinzierl. SYCL compute kernels for ExaHyPE, 2023.

[10] Uzmar Gomez, Gonzalo Brito Gadeschi, and Tobias Weinzierl. GPU Offloading in ExaHyPE Through C++ Standard Algorithms, 2023.

[11] Tobias Weinzierl. The Peano Software—Parallel, Automaton-Based, Dynamically Adaptive Grid Traversals. *ACM Trans. Math. Softw.*, 45(2), 2019.

[12] J.G. Zhou, D.M. Causon, C.G. Mingham, and D.M. Ingram. The Surface Gradient Method for the Treatment of Source Terms in the Shallow-Water Equations. *Journal of Computational Physics*, 168(1):1–25, 2001.

[13] Han Zhang, Tobias Weinzierl, Holger Schulz, and Baojiu Li. Spherical accretion of collisional gas in modified gravity I: self-similar solutions and a new cosmological hydrodynamical code. *Monthly Notices of the Royal Astronomical Society*, 515(2):2464–2482, 2022.

[14] Zihua Niu, Alice-Agnes Gabriel, Linus Seelinger, and Heiner Igel. Modeling and Quantifying Parameter Uncertainty of Co-seismic Non-classical Nonlinearity in Rocks, 2023.

[15] Linus Seelinger, Anne Reinarz, Leonhard Rannabauer, Michael Bader, Peter Bastian, and Robert Scheichl. High Performance Uncertainty Quantification with Parallelized Multilevel Markov Chain Monte Carlo. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[16] Lukas Krenz. Cloud Simulation with the ExaHyPE-Engine. Master's thesis, Technische Universität München, 2019.

[17] Sven Hingst. Adjoint-Guided Mesh Refinement for Earthquake Simulations. Bachelor's thesis, Technische Universität München, 2022.

[18] Eleuterio Toro. *Shock-Capturing Methods for Free-Surface Shallow Flows*. 2001.

[19] D Ambrosi. Approximation of shallow water equations by Roe's Riemann solver. *International journal for numerical methods in fluids*, 20(2):157–168, 1995.

[20] Manuel Jesús Castro Díaz, Alberto Pardo, Carlos Parés Madroñal, and Eleuterio F. Toro. On some fast well-balanced first order solvers for nonconservative systems. *Math. Comput.*, 79:1427–1472, 2009.

[21] Derek S. Bale, Randall J. LeVeque, Sorin Mitran, and James A. Rossmanith. A Wave Propagation Method for Conservation Laws and Balance Laws with Spatially Varying Flux Functions. *SIAM Journal on Scientific Computing*, 24(3):955–978, 2003.

[22] Kyle T Mandli, Aron J Ahmadia, Marsha Berger, Donna Calhoun, David L George, Yiannis Hadjimichael, David I Ketcheson, Grady I Lemoine, and Randall J LeVeque. Clawpack: building an open source ecosystem for solving hyperbolic PDEs. *PeerJ Computer Science*, 2:e68, 2016.

[23] Leonhard Rannabauer, Stefan Haas, Dominic Etienne Charrier, Tobias Weinzierl, and Michael Bader. Simulation of tsunamis with the exascale hyperbolic PDE engine ExaHyPE. In *Environmental Informatics: Techniques and Trends. Adjunct Proceedings of the 32nd edition of the EnviroInfo.*, 2018.

[24] Michael Dumbser and Dinshaw S. Balsara. A New Efficient Formulation of the HLLEM Riemann Solver for General Conservative and Non-Conservative Hyperbolic Systems. *J. Comput. Phys.*, 304:275–319, 2016.

[25] Brisa N. Davis and Randall J. LeVeque. Adjoint Methods for Guiding Adaptive Mesh Refinement in Tsunami Modeling. *Global Tsunami Science: Past and Future, Volume I*, pages 4055–4074, 2017.

[26] Brisa N. Davis and Randall J. LeVeque. Analysis and Performance Evaluation of Adjoint-Guided Adaptive Mesh Refinement for Linear Hyperbolic PDEs Using Clawpack. *ACM Trans. Math. Softw.*, 46(3), 2020.

[27] D. Bale, R. J. LeVeque, S. Mitran, and J. A. Rossmanith. A wave-propagation method for conservation laws and balance laws with spatially varying flux functions. *SIAM J. Sci. Comput.*, 24:955–978, 2002.

[28] Randall J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002.

[29] Frank I. Gonzalez, Randall J. LeVeque, Loyce M. Adams, Chris Goldfinger, George R. Priest, and Kelin Wang. Probabilistic Tsunami Hazard Assessment (PTHA) for Crescent City, CA. Technical report, 2014.

[30] K. Goda, T. Yasuda, N. Mori, A. Muhammad, R. De Risi, and F. De Luca. Uncertainty quantification of tsunami inundation in Kuroshio, Kochi Prefecture, Japan, using the Nankai–Tonankai megathrust rupture scenarios. *Natural Hazards and Earth System Sciences*, 20(11):3039–3056, 2020.

[31] Ihab Sraj, Kyle T. Mandli, Omar M. Knio, Clint N. Dawson, and Ibrahim Hoteit. Uncertainty quantification and inference of Manning's friction coefficients using DART buoy data during the Tōhoku tsunami. *Ocean Modelling*, 83:82–97, 2014.

[32] Monika Johanna Kreitmair. *The Effect of Uncertainty on Tidal Stream Energy Resource Estimates*. Springer Nature, 2020.

[33] Linus Seelinger, Anne Reinarz, Jean Benezech, Mikkel Bue Lykkegaard, Lorenzo Tamellini, and Robert Scheichl. Lowering the Entry Bar to HPC-Scale Uncertainty Quantification, 2023.

A. Python Implementation

The SWE implementation itself is defined by a main script - *swe.py* - that deals with different configuration options and partially also with the parsing of input files.

The script accepts arguments for:

- The type of solver to be used. Currently, ExaHyPE 2's working solvers are a provided FV Rusanov solver and user-defined FV solvers, but arguments for the Runge-Kutta Discontinuous Galerkin solver are also already provided.
- Various configurable parameters of the simulation, such as the end time, patch size, maximum cell size, and the interval at which output is to be saved to files, for both the simulation as a whole and possible probes within the domain.
- AMR: refinement levels and refinement tolerance.
- The scenario to be simulated. For non-standardized scenarios, the dimensions of the domain can be specified as well.
- A plethora of netCDF related arguments, such as file paths and keys. There are also two arguments related to the maximum cell size when using netCDF and AMR. When switched on, *swe.py* will either attempt to import the `netCDF4` package to read in the resolution of the input data and adjust the maximum cell size to fit the finest input resolution, or a data resolution according to which to set the maximum cell size can be supplied.
- Various options related to guided AMR. These include providing a pre-computed solution, or setting a spatial and temporal area of interest, as well as different other aspects of the adjoint problem's simulation. One notable option is to only solve the adjoint problem and run no follow-up simulation in ExaHyPE 2.
- Two options related to UQ, which move the origin of the earthquake to given coordinates, or add probes to the domain.

The solver is a part of the whole Peano project generated in *swe.py*. The code, which describes how the solver is supposed to solve the SWE, is detailed in various classes with clear delegation of functionality to different classes (cf. Figure A.1)[1].

[1] Again, a detailed artifact description can be found under `https://doi.org/10.5281/zenodo.8305725`.

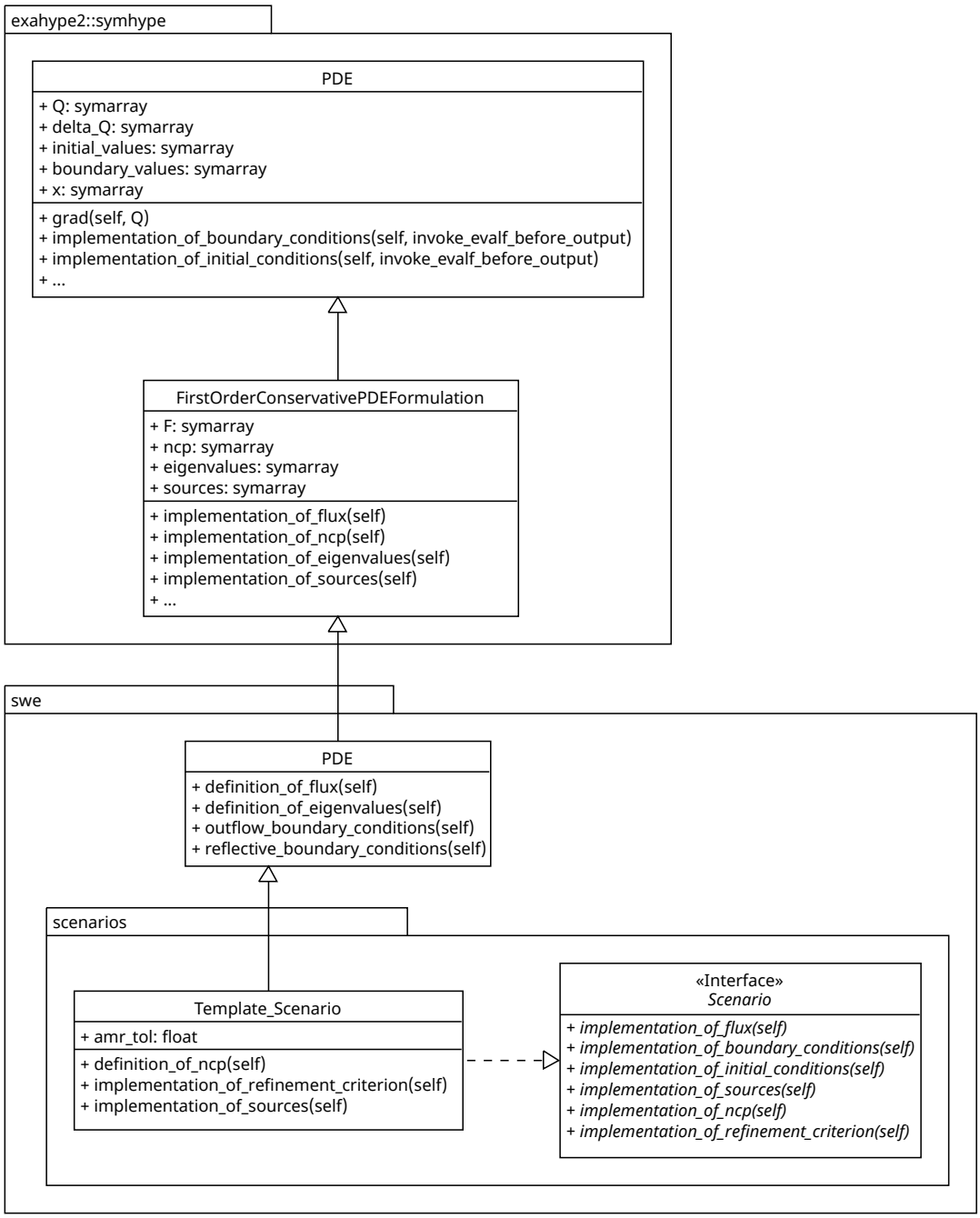


Figure A.1.: UML class diagram of `Template_Scenario`'s inheritance hierarchy.

The interface `Scenario` defines the possible overridable methods of each scenario, thus explicitly stating the customization options of the shallow-water application usable when defining a scenario. The class `PDE` in the `swe` module of the SWE application contains the definitions of both the flux and the eigenvalues (cf. Code Listing A.1).

```
def definiton_of_flux(self):
    # Flux [unknowns, dimensions]
    self.F[0, 0] = self.hu
    self.F[1, 0] = self.hu * self.u + (self.g * self.h**2)/2
    self.F[2, 0] = self.h * self.u * self.v

    self.F[0, 1] = self.hv
    self.F[1, 1] = self.h * self.u * self.v
    self.F[2, 1] = self.hv * self.v + (self.g * self.h**2)/2

def definition_of_eigenvalues(self):
    # Eigenvalues [unknowns, dimensions]
    self.eigenvalues[0, 0] = self.u
    self.eigenvalues[0, 1] = self.v

    self.eigenvalues[1, 0] = self.u + sympy.sqrt(self.g * self.h)
    self.eigenvalues[1, 1] = self.v + sympy.sqrt(self.g * self.h)

    self.eigenvalues[2, 0] = self.u - sympy.sqrt(self.g * self.h)
    self.eigenvalues[2, 1] = self.v - sympy.sqrt(self.g * self.h)
```

Code Listing A.1: Flux and eigenvalues in class `PDE`.

These definitions closely follow the definition of the SWE as they are given in Section 2.2 and demonstrate quite well exactly how easy setting up ExaHyPE 2 is.

For convenience purposes, `PDE` also provides outflow and reflective boundary conditions (cf. Code Listing A.2), as `PDE` is extended by `Template_Scenario`, which is itself extended by every scenario provided in the SWE application.

```
def outflow_boundary_conditions(self):
    # outflow boundary conditions
    self.boundary_values[0] = self.h
    self.boundary_values[1] = self.hu
    self.boundary_values[2] = self.hv
    self.boundary_values[3] = self.b

def reflective_boundary_conditions(self):
    # reflective boundary conditions
    self.boundary_values[0] = self.h
    self.boundary_values[1] = -self.hu
    self.boundary_values[2] = -self.hv
    self.boundary_values[3] = self.b
```

Code Listing A.2: Outflow and reflective boundary conditions in class `PDE`.

As each scenario should extend both `PDE` and `Scenario`, most implementations remain the same for the different scenarios. That is why `Template_Scenario` bundles together various calls to and definitions of methods, so that subsequent scenarios only have to define the initial values if they do not have special requirements. `Template_Scenario` uses outflow boundary conditions provided in `PDE` by default and sets `implementation_of_sources()` to `return None_Implementation`, as `Template_Scenario` provides an implementation of the source terms given in Section 2.2 via a non-conservative product. In the case that the user wishes to change this behavior, both `implementation_of_sources()` and `implementation_of_ncp()` can be simply overridden. This might be done in order to include more sophisticated source terms, such as a friction term.

The implementation of the source term via the non-conservative product is given in Code Listing A.3.

```
def definition_of_ncp(self):
    grad_b = self.grad(self.b)

    # NCP [unknowns, dimensions]
    self.ncp[0, 0] = 0.0
    self.ncp[1, 0] = self.g * self.h * grad_b
    self.ncp[2, 0] = 0.0

    self.ncp[0, 1] = 0.0
    self.ncp[1, 1] = 0.0
    self.ncp[2, 1] = self.g * self.h * grad_b
```

Code Listing A.3: Non-conservative product in class `Template_Scenario`.

With `grad_b` being equivalent to b_x or b_y in the source term for x- and y-dimension respectively.

`implementation_of_refinement_criterion()` is one important exception to code generation, as this is most practically defined in C++ directly. `Template_Scenario` also provides a default implementation for this method in the form of the in Section 4.1 already discussed surface-flagging approach. All given implemented scenarios extend only `Template_Scenario` as it extends both `PDE` and `Scenario` and provides useful default implementations that only have to be changed by a few scenarios.

B. C++ Implementation

The SWE applications consists of mainly three C++ classes, that are injected into the solver under various circumstances, as seen in Figure 3.2. These classes are:

- `TopologyParser`: Configurable input file parser for different use-cases. This class does most of the work for the `Tsunami` scenario.
- `SWEAdjoint`: Solves the adjoint problem using the one-dimensional solver detailed in Algorithm 6 and the two-dimensional scheme in Algorithm 7. It uses a configured TopologyParser to read in the bathymetry.
- `AdjointParser`: Reads in the solution file of an adjoint problem. Further, it provides the maximum inner product for guided AMR as detailed in Algorithm 8.

Both `SWEAdjoint` and `AdjointParser` are more explained in detail in Section 4.2. This section will only focus on `TopologyParser` and some supporting code.

Topology Parser The `TopologyParser` class offers two constructors. One is used only by the adjoint problem in the `SWEAdjoint` class. This constructor takes only a file path for the bathymetry file and three optional keys for the x- and y-dimensions and the bathymetry in the netCDF input file. The second constructor offered takes both a bathymetry and a displacement input file, as well as optional keys for both files. It also takes optional coordinates, which point to a new origin of the displacement within the bathymetry.

When instantiated, the `TopologyParser` object immediately reads in the files provided using the `parse_bathymetry_file()` and `parse_displacement_file()` functions. In the event of an error occurring during file I/O, the `TopologyParser` returns from its constructor and prints an error message on the standard output. The information required by the `TopologyParser` consists only of the x-coordinates, y-coordinates, and the values of the variable stored within the file, which are indexed by the x- and y-coordinates. Next to storing the x-y-indexed values, it stores the extreme values of the coordinates of both bathymetry and, if applicable, displacement file. The displacement file's absolute extreme values have to be strictly less or equal to the bathymetry's absolute extreme values, so that the displacement is entirely contained within the bathymetry. Otherwise, a correct functioning of `TopologyParser` can not be guaranteed.

The functions `sample_bathymetry(x,y)` and `sample_displacement(x,y)` use two interpolation functions to map the simulation domain to the target netCDF values. The first interpolation maps the simulation domain to the coordinate space of the respective netCDF file. The behavior of this function is described by

$$p_1(z, z_{max}, z_{bath}^{min}, z_{bath}^{max}) = \frac{z_{bath}^{max} - z_{bath}^{min}}{z_{max}} \cdot z + z_{bath}^{min}, \tag{B.1}$$

where z is a coordinate in the x- or y-dimension within the simulation domain, z_{max} the maximum value of z, as the minimum value for z is assumed to be 0, and z_{bath}^{min} and z_{bath}^{max} the minimum and

maximum coordinates in the dimension of z within the bathymetry file. p_1 is applied to both x and y, and the resulting coordinates can then be compared to various criteria. In `TopologyParser`, the coordinates are only checked for validity. Coordinates are considered valid if they are within their respective extreme values. If they are not valid, either a bathymetry of 20 or a displacement of 0 is returned. If the coordinates are valid, then a second interpolation maps the netCDF coordinates onto the underlying one-dimensional array. The second interpolation function is given by:

$$p_z(z, z_{cdf}^{min}, z_{cdf}^{max}, dim_{cdf}^{z}) = \left\lfloor \frac{dim_{cdf}^{z} - 1}{z_{cdf}^{max} - z_{cdf}^{min}} \cdot z + \frac{z_{cdf}^{min} \cdot dim_{cdf}^{z}}{z_{cdf}^{min} - z_{cdf}^{max}} \right\rfloor, \tag{B.2}$$

$$p_2(\bar{x}, \bar{y}, x_{cdf}^{min}, y_{cdf}^{min}, x_{cdf}^{max}, y_{cdf}^{max}, dim_{cdf}^{x}, dim_{cdf}^{y}) = p_z(\bar{y}, y_{cdf}^{min}, y_{cdf}^{max}, dim_{cdf}^{y}) \cdot dim_{cdf}^{x} \tag{B.3}$$

$$+ p_z(\bar{x}, x_{cdf}^{min}, x_{cdf}^{max}, dim_{cdf}^{x}). \tag{B.4}$$

The function p_z is used to individually interpolate coordinates onto array indices. The new variables introduced for p_2 are dim_{cdf}^{x} and dim_{cdf}^{y}, the array length of the corresponding dimension. The subscript *cdf* indicate that both bathymetry or displacement limits can be used here, depending on what function the interpolation is called in. The interpolated netCDF coordinates from p_1 are inserted into p_2 as $\bar{x}$ and $\bar{y}$.

If the displacement is to be moved for UQ, then the netCDF coordinates resulting from p_1 are transformed as $\bar{z}_{trans} = \bar{z} - z_{origin} - 0.5(z_{displ}^{max} + z_{displ}^{min})$, with the *displ* subscript denoting that these values belong to the displacement file's coordinate system. The transformed coordinates are then checked for validity and used in p_2.

NetCDF Reader and Helper The `TopologyParser` class uses the functions provided within the `parser::NetCDFReader` and `parser::NetCDFHelper` namespaces extensively. The functions in `parser::NetCDFReader` provide convenient wrappers for the standard netCDF library functions, such as opening/closing netCDF files, retrieving global attributes of various types, and reading variable values into 1D, 2D, or 3D arrays.

The functions in `parser:NetCDFHelper` implement the p_1 and p_2 interpolation functions as `transformIndexSimulationToCDFRange()` and `transformIndexCDFRangeToArray()`.

NetCDF Writer This class is a wrapper for netCDF library functions as well. Realized as a class, the `NetCDFWriter` saves the netCDF IDs of the variables it has to write, as well as the dimensions of the variables. It is only used in the `SWEAdjoint` class, as Peano does not support netCDF output yet, but the solution to the adjoint problem is easiest to read in the netCDF format.

In its constructor, `NetCDFWriter` takes an output file path, the bathymetry file path, the sizes of the x- and y-dimensions, a pointer to the bathymetry array read in by `TopologyParser` earlier, the coordinates of the area of interest, and the start and end time for the guided AMR. Most of these parameters are simply written into the resulting netCDF file as attributes, so that later users can reconstruct the parameters used to run the simulation. The only parameter written to the file as a variable is the bathymetry array, as it is time-independent and thus only needs to be written once before the simulation starts.

The time-dependent variables are written using the `writeTimeStep()` function. As `SWEAdjoint` has all the simulation data in one-dimensional, column wise arrays, `writeTimeStep()` simply

iterates through the columns and writes them to the output file. Concerning the time variable, it writes the parameterized t value at the current checkpoint position, which is also parameterized.