SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Coupling of Tsunami Simulations in ExaHyPE 2 with the UM-Bridge Client-Server Model

Emin Mert Sunacoglu

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Coupling of Tsunami Simulations in ExaHyPE 2 with the UM-Bridge Client-Server Model

Author:	Emin Mert Sunacoglu
Supervisor:	Prof. Dr. Michael Bader
Advisor:	Mario Wille
Submission Date:	15.08.2024

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

Munich, 15.08.2024 Emin Mert Sunacoglu

Acknowledgments

I want to express my deepest gratitude to my advisor, Mario Wille, for providing me the opportunity to work on this project and for providing me with timely guidance and support during the thesis.

I would like to thank Prof. Dr. Michael Bader for supervising my thesis.

I also would like to thank Dr. Anne Reinarz for her clarifications regarding her paper and its implementations, which were instrumental during the research.

I am also grateful to my family and friends for their constant encouragement and understanding throughout this journey.

Abstract

Tsunamis as one of the most destructive natural disasters, have driven extensive research. This thesis focuses on the development of a reliable solver for the shallow water equations (SWEs) used for describing tsunamis, using the ExaHyPE 2 engine, and implementing an uncertainty quantification framework for tsunami models.

The primary objectives of this research are to accurately model the 2011 Tōhoku tsunami and estimate its initial conditions using the measurement data from real-world buoys. The solver developed in ExaHyPE 2 showed accurate results in modeling the Tōhoku tsunami. Furthermore, the integration of UM-Bridge allowed the application of the Metropolis-Hastings Markov Chain Monte Carlo (MHMCMC) algorithm to estimate the initial displacement data with reasonable precision.

Despite the promising results, some simplifications in the modeling process, such as using larger computational cells, introduced minor inaccuracies. These inaccuracies, while not significantly affecting the overall results, are important to note for future improvements in the model.

This thesis contributes to the field of tsunami modeling by providing a scalable and accurate model for tsunamis using ExaHyPE 2.

Keywords: Tsunami simulation, ExaHyPE 2, Uncertainty Quantification, UM-Bridge

Contents

Acknowledgments **iii**

Abstract **iv**

1. Introduction **1**

2. Background **2**

2.1. Partial Differential Equations 2

2.2. Shallow Water Equations 2

2.3. Inverse Problems 4

2.4. Uncertainty Quantification 4

3. Related Work **5**

3.1. Peano 4 5

3.2. ExaHyPE 2 6

4. Implementation of the Shallow Water Equations in ExaHyPE 2 **7**

4.1. Finite Volume Method 7

4.1.1. Generic Riemann Solver 8

4.1.2. Custom Riemann Solvers 8

4.2. ADER-DG 13

4.3. Limiting 15

4.4. Particles 15

5. UM-Bridge in ExaHyPE 2 **17**

5.1. UM-Bridge 17

5.2. Monte Carlo Algorithms 20

5.2.1. Monte Carlo 20

5.2.2. Markov Chain Monte Carlo 20

5.2.3. Metropolis Hastings Markov Chain Monte Carlo 20

5.2.4. Multilevel Markov Chain Monte Carlo 21

6. Results **23**
6.1. Evaluation of the Solvers 23
6.1.1. Radial Dam Break Scenario 23
6.1.2. Tōhoku Tsunami 33
6.2. UM-Bridge with Metropolis-Hastings Markov Chain Monte Carlo . . . 42

7. Conclusion and Future Work **45**

List of Figures **46**

List of Tables **48**

Code listings **49**

Bibliography **50**

A. Implementation of SWE in ExaHyPE 2 **53**

B. Implementation of UM-Bridge in ExaHyPE 2 **57**

1. Introduction

Tsunamis are among the most destructive natural phenomena, causing extensive damage and loss of life when they strike coastal areas. The 2011 Tōhoku tsunami, despite not being the largest on record, resulted in a significant nuclear accident, which caused the tragic loss of over 18,000 lives, highlighting the impact such events can have. Therefore, the accurate modeling and prediction of tsunamis are important, driving extensive research in this area.

Partial Differential Equations (PDEs) are fundamental in simulating physical processes, providing a robust framework to model the dynamics of systems over time and space. The shallow water equations (SWEs) describe the water flow. Solving these equations is essential for accurately modeling and predicting the behavior of a given tsunami.

However, accurately modeling tsunamis is not without challenges. Uncertainty quantification (UQ) is pivotal in validating the complex models used in these simulations. UQ helps assess how variations in input parameters influence model outputs or estimate unknown parameters based on observed data, thereby enhancing the reliability and robustness of the simulations. Because UQ algorithms are broadly applicable across various domains, it is essential to have a tool to couple these algorithms with different simulation models.

This thesis leverages the ExaHyPE 2 framework (**Exa**scale **Hy**perbolic **P**DE **E**ngine) to develop a solver for the shallow water equations. This framework, known for its high-performance computing capabilities, is crucial for accurately modeling tsunami dynamics. Additionally, the thesis implements a client-server framework for applying UQ algorithms, focusing on estimating the initial parameters for the 2011 Tōhoku tsunami using real-world measurements.

Chapter 2 provides the necessary background information relevant to the study. In Chapter 3, the ExaHyPE 2 framework is introduced. Chapter 4 introduces three solvers developed using the ExaHyPE 2 framework that is used to model tsunamis, and Chapter 5 describes the UM-Bridge (**u**ncertainty quantification and **m**odeling **bridge**) framework that is implemented to couple our model with UQ algorithms. Chapter 6 evaluates the accuracy of the solvers and evaluates the results of the UQ algorithms that are implemented in Chapter 5. Finally, Chapter 7 summarized the findings, and suggests areas of future research and improvement.

2. Background

2.1. Partial Differential Equations

Partial Differential Equations (PDEs) are widely used in simulations because they effectively describe how physical systems change over time and space. Whether modeling the spread of a tsunami, the flow of heat, or the movement of air, PDEs capture these processes' continuous and dynamic nature, making them crucial for accurate predictions in various scientific and engineering applications.

PDEs involve partial derivatives of an unknown function of two or more parameters [8]. In general, equations in a k-th order system of partial differential equations where F is a function over the unknown function u have the form:

$$F(D^k u(x), D^{k-1} u(x), ..., u(x), x) = 0 \tag{2.1}$$

Where F is some function with

$$F : \mathbb{R}^{n^k} \times \mathbb{R}^{n^{k-1}} \times .... \times \mathbb{R}^n \times \mathbb{R} \times U \to \mathbb{R} \tag{2.2}$$

and D is the derivative of the function u over one of its parameters.

2.2. Shallow Water Equations

The system of PDEs that are used to describe tsunami scenarios is called shallow water equations. The shallow water equations (SWEs) are derived from more complicated three-dimensional Navier–Stokes equations [18]. The vertical scale of the domain for a tsunami simulation is much greater than the horizontal scale; therefore, the domain and the problem can be reduced to two dimensions, taking the vertical water height and bathymetry as input, resulting in 2-D SWEs.

Despite the name, SWEs don't require the fluid to be shallow as they can be used for describing a tsunami wave in the ocean, where the depth can reach 5 km [12]. The following chapters use these equations to simulate artificial scenarios and the real-world 2011 Tōhoku tsunami.

SWEs combine the laws of conservation of mass and momentum [12]:

Mass Conservation:

$$\frac{\partial h}{\partial t} + \frac{\partial (hu)}{\partial x} + \frac{\partial (hv)}{\partial y} = 0$$

where h is the water depth, and u and v are the depth-averaged velocities in the x- and y-directions, respectively.

Momentum Conservation:

$$\frac{\partial (hu)}{\partial t} + \frac{\partial}{\partial x}\left(hu^2 + \frac{1}{2}gh^2\right) + \frac{\partial (huv)}{\partial y} = -gh\frac{\partial b}{\partial x}$$

$$\frac{\partial (hv)}{\partial t} + \frac{\partial (huv)}{\partial x} + \frac{\partial}{\partial y}\left(hv^2 + \frac{1}{2}gh^2\right) = -gh\frac{\partial b}{\partial y}$$

g is the acceleration due to gravity and b is the bathymetry.

To use ExaHyPE 2, shallow water equations need to be rewritten in the following form [18]:

$$\frac{\partial}{\partial t}\begin{pmatrix} h \\ hu \\ hv \\ b \end{pmatrix} + \nabla \cdot \begin{pmatrix} hu & hv \\ hu^2 & huv \\ huv & hv^2 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 \\ hg\frac{\partial (b+h)}{\partial x} \\ hg\frac{\partial (b+h)}{\partial y} \\ 0 \end{pmatrix} = 0 \tag{2.3}$$

$$\begin{pmatrix} h \\ hu \\ hv \\ b \end{pmatrix} = Q, \quad \begin{pmatrix} hu & hv \\ hu^2 & huv \\ huv & hv^2 \\ 0 & 0 \end{pmatrix} = F(Q), \quad \begin{pmatrix} 0 \\ hg\frac{\partial (b+h)}{\partial x} \\ hg\frac{\partial (b+h)}{\partial y} \\ 0 \end{pmatrix} = B(Q)\nabla Q$$

where h is the height of the water, u and v are the velocities in the x- and y-direction, respectively, and b is the bathymetry and g is the earth's gravitational acceleration.

The eigenvalues that are used for the timestep calculations in both x- and y-directions are given by

$$\begin{aligned} \lambda_{x1} = u, \lambda_{x2} &= v + \sqrt{gh}, \lambda_{x3} = v - \sqrt{gh}, \\ \lambda_{y1} = v, \lambda_{y2} &= v + \sqrt{gh}, \lambda_{y3} = v - \sqrt{gh} \end{aligned} \tag{2.4}$$

2.3. Inverse Problems

Inverse problems aim to estimate the unknown input parameters of model equations using observations [27].

When dealing with complex problems, such as shallow water equations, it is not feasible to invert the problem directly due to complexity and computational difficulties. This thesis employs the Bayesian approach to solve the inverse problem. The Bayesian approach attempts to determine the distribution of parameters from available data instead of inverting it [4].

In our use case, we aim to obtain the location of the seabed displacement leading to the 2011 Tōhoku tsunami from the data of two available DART buoys near the Japanese coast [1].

We define our model as a function as a function G of the displacement θ and our observation of the outcome as y:

$$y = G(\theta, Q) + e, \tag{2.5}$$

where e is the noise in the observations.

We aim to calculate the initial data from our output function. Since our problem is too complex to solve analytically, a common approach is to define a cost function that depends on the error between the solution of the model and the outcome. We treat the problem as a minimization problem for our cost function. The simplest way to define our cost function would be:

$$\theta = \arg\min_{\theta} ||y - G(\theta, Q)|| \tag{2.6}$$

2.4. Uncertainty Quantification

Uncertainty quantification (UQ) is a field dedicated to understanding, characterizing, and managing the uncertainties present in models. It involves identifying sources of uncertainty, such as variability in input parameters, model approximations, data quality, and external factors. UQ quantifies the extent and impact of these uncertainties using statistical methods to measure how they propagate through models and affect outcomes.

A common problem in UQ is finding the uncertain input parameters in a given model from real-world measurements, hence solving an inverse problem.

[1] The data for DART buoys was obtained from NDBC https://www.ndbc.noaa.gov/.

3. Related Work

3.1. Peano 4

ExaHyPE (**Exa**scale **Hy**perbolic **P**DE **E**ngine) is part of the Peano project, a framework for solvers operating on dynamically adaptive Cartesian meshes.

Peano primarily focuses on mesh management, data storage, distribution, and traversal of the grids, serving as the foundation for various engines and toolboxes tailored to different application areas. The latest iteration, Peano 4, is the fourth generation of this framework and is available as open-source software [25].

In computational simulations, the efficiency and accuracy of the numerical solution often rely on the underlying grid structure and the employed load-balancing mechanisms. Peano provides a robust framework for dynamically adaptive Cartesian meshes, ensuring optimal performance and scalability.

In ExaHyPE, the grid partition and load balancing are handled by the Peano framework. Peano allows users to increase the accuracy of the simulation in certain areas by using adaptive mesh refinement (AMR) when necessary [18]. This can improve performance and accuracy in tsunami simulations as the wave usually only occupies a small portion of the domain. The rest is typically static but still needs to be calculated and can be equally computationally intensive [13], even though it results in no net updates. By refining the cells in the critical domain, we can increase the accuracy of our simulations while suffering a minimal performance loss.

Grid initialization in Peano 4 involves setting up the initial domain and defining the refinement criteria. The framework manages grid traversal and updates, ensuring the mesh evolves appropriately throughout the simulation. Efficient data structures and algorithms are employed to handle the dynamically changing mesh.

In parallel computing environments, load balancing is crucial for maximizing resource utilization and minimizing computational time. An unbalanced load can lead to some processors being idle while others are overloaded, resulting in inefficiencies. Peano 4 incorporates several load balancing strategies, like static and dynamic load balancing, to distribute the computational workload evenly across processors.

3.2. ExaHyPE 2

To be able to work with uncertainty quantification on tsunami simulations, we first need to implement a method to solve the forward problem and simulate the tsunami resulting from a given initial condition. The forward problem, in the context of tsunami simulations, involves predicting the wave's evolution over time, given the seafloor's initial bathymetry and the displacement caused by an earthquake. This requires solving the equations described in Chapter 2.2, where the solution represents the physical behavior of the tsunami.

ExaHyPE 2 is an open-source software engine that can solve first-order linear and non-linear hyperbolic partial differential equations in the following form [18]:

$$\frac{\partial}{\partial t}Q + \nabla F(Q, \nabla Q) + B(Q) \cdot \nabla Q = S(Q) + \sum_{i=1}^{n_{ps}} \delta_i \tag{3.1}$$

where Q stands for the state vector, $F(Q)$ is the flux tensor, $B_i(Q)$ is the non-conservative product (often shortened to NCP), $S(Q)$ is the source term of the equation and $\sum_{i=1}^{n_{ps}} \delta_i$ denotes point sources.

PDEs that are derived from conservation laws in physics have a wide range of application areas like uncertainty quantification applications [16, 22], and elastodynamics [18].

Due to the scalability of ExaHyPE 2, these simulations can be run on a regular laptop or utilize the resources of supercomputers.

While the predecessor of the ExaHyPE 2 also provided a way to solve these equations, ExaHyPE 2 provides a better framework to utilize resources of the supercomputers while solving the related PDEs using MPI and OpenMP [19] parallelization, as well as support for GPU offloading [26, 9, 14].

ExaHyPE 2 and Peano handle the glue code and time and space discretization, allowing users to develop simulation codes that would take a long time to create from scratch.

To describe a specific problem, the user needs 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 δ_i and eigenvalues have to be specified for all methods that use adaptive time stepping. The user can also redefine the Riemann solver to describe the change of the source terms [18].

4. Implementation of the Shallow Water Equations in ExaHyPE 2

There are multiple ways to solve the shallow water equations (c.f. Eq. 2.3) in order to simulate tsunamis. They vary in how the domains are represented and the discretization of time. In the context of this thesis, we use adaptive time stepping in all of our simulations, which means that the time steps are calculated on every iteration using the maximal eigenvalues, thus preventing the waves from skipping cells or overlapping with another wave. In Sections 4.1 and 4.2, we introduce two separate ways to represent the cells in the domain and implement solvers for solving the Equation 2.3.

4.1. Finite Volume Method

In finite volume methods, the computational domain is discretized into small, non-overlapping control volumes or cells. Its cell-averaged value represents the state vector Q, assuming a uniform distribution within each cell. The partial differential equations are integrated over each control volume, converting volume integrals containing divergence terms into surface integrals by applying the divergence theorem [3]. Then, the flows are calculated by solving a series of Riemann problems on the boundary of each subdomain [7]. SWEs are based on integrating the equation from 2.3. Thus,

$$\int_V \left(\frac{\partial \mathbf{Q}}{\partial t} + (\nabla \cdot \mathbf{F}(\mathbf{Q}))^T \right) dV = \int_V \mathbf{B}(\mathbf{Q}) dV \tag{4.1}$$

The flux integral $\nabla \cdot \mathbf{F}(\mathbf{Q}))^T$ is approximated using numerical flux functions at the cell interfaces. For approximating methods like Rusanov, e.g. 4.1.1, FWaves e.g. 4.1.2 or HLL fluxes, e.g. 4.1.2 can be used.

The integral for the state vector is then calculated using the Equation 4.1, thus giving us the formula:

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x}(F_{i+1/2}^n - F_{i-1/2}^n) + \Delta t S_i^n$$

4.1.1. Generic Riemann Solver

ExaHyPE 2 provides a generic Rusanov FV solver that can solve artificial scenarios that do not involve dry cells. In Chapter 6.1.1, we will use this solver as a baseline for testing our other solver implementations. It provides approximate results in simple scenarios but is insufficient to simulate real-world tsunamis.

4.1.2. Custom Riemann Solvers

The generic Riemann solver is not precise enough and cannot handle cases involving dry cells. Therefore, we implement a custom Riemann solver by describing the in- and outgoing fluxes $A^{\pm\Delta Q}$ over time.

FWave Solver

In the open ocean, a tsunami has a small wave height compared to the ocean's depth and a long wavelength. Because of this, its movement can be considered almost linear, with variations caused by changes in bathymetry. As a tsunami approaches shore, however, the amplitude typically increases while the depth of the water decreases, and nonlinear effects become important [13]. The FWave solver focuses on the open ocean and ignores the nonlinear effects of the SWEs.

The FWave solver is designed for the one-dimensional shallow water equations. The initial idea is to solve the Riemann problem in the first dimension and then do an x- and y-sweep of the domain to extend the solver to two dimensions.

In one space dimension the equation has the form [2]:

$$\frac{\partial \mathbf{Q}}{\partial t} + f(q, x) = 0 \tag{4.2}$$

Before defining the Riemann solver, we must implement the fluxes defined in Equation 2.3. We inherit the non-conservative products into the flux function, thus eliminating the non-conservative products and defining the fluxes as in the Algorithm 1.

To calculate the time step Δt, we need to prevent waves from interacting with each other, meaning they should not cross the cell center x_i at any given timestep, ensuring two incoming waves from opposite directions cannot meet. This condition is satisfied by using a timestep relaxation factor smaller than $1/2$ and calculating the timestep as:

$$\Delta t < \frac{1}{2} \cdot \frac{\Delta x}{\lambda^{max}}$$

where λ are the eigenvalues which are defined as in Algorithm 2, and for the timestep relaxation factor (TSR) $TSR = 0.4 < \frac{1}{2}$ is chosen.

Then, we calculate the net updates for left and right going waves, thus implementing our Riemann solver as in the Algorithm 3.

Algorithm 1 Flux Function

```
Input: $\mathbf{q}$: state vector with elements $h, hu, hv, b$
       $n \in \{x, y\}$: normal direction
Output: $\mathbf{F}$: array containing flux in normal direction
1: function FLUX($\mathbf{q}, n$)
2:     if $n = x$ then
3:         $\mathbf{F} \leftarrow \begin{pmatrix} hu \\ h(u^2) + \frac{1}{2}gh^2 \\ huv \\ 0.0 \end{pmatrix}$
4:     else
5:         $\mathbf{F} \leftarrow \begin{pmatrix} hv \\ huv \\ h(v^2) + \frac{1}{2}gh^2 \\ 0.0 \end{pmatrix}$
6:     end if
7:     return $\mathbf{F}$
8: end function
```

Algorithm 2 Eigenvalues

```
Input: q: state vector with elements $h, hu, hv, b$
       $n \in \{x, y\}$: normal direction
Output: $\lambda$: array containing eigenvalues
1: function EIGENVALUE($\mathbf{q}, n$)
2:     if $n = x$ then
3:         $u_n \leftarrow \frac{\sqrt{2}u}{h^2}$
4:     else
5:         $u_n \leftarrow \frac{\sqrt{2}v}{h^2}$
6:     end if
7:     $\lambda \leftarrow \begin{pmatrix} u_n + \sqrt{gh} \\ u_n - \sqrt{gh} \\ u_n \\ 0.0 \end{pmatrix}$
8:     return $\lambda$
9: end function
```

Algorithm 3 FWave Riemann Solver

```
1: function RIEMANN($\mathbf{qL}, \mathbf{qR}, \mathbf{FL}, \mathbf{FR}$)
2:     $h^{Roe} \leftarrow \frac{1}{2}(h_l + h_r)$
3:     $u^{Roe} \leftarrow \frac{u_l\sqrt{h_l}+u_r\sqrt{h_r}}{\sqrt{h_l}+\sqrt{h_r}}$
4:     $\lambda^{Roe} \leftarrow \begin{pmatrix} u^{Roe} - \sqrt{gh^{Roe}} \\ u^{Roe} + \sqrt{gh^{Roe}} \end{pmatrix}$
5:     $\Delta x \Psi_{i-1/2} \begin{pmatrix} 0 \\ \frac{-g(b_r-b_l)h^{Roe}}{2} \end{pmatrix}$
6:     $\Delta F \leftarrow FR - FL - \Delta x \Psi_{i-1/2}$
7:     $A^{-\Delta Q} \leftarrow \sum_{p:\lambda^{Roe}<0} \Delta F$
8:     $A^{+\Delta Q} \leftarrow \sum_{p:\lambda^{Roe}>0} \Delta F$
9:     return $max(\lambda^{Roe}[0], \lambda^{Roe}[1])$
10: end function
```

Disadvantages of the FWave Solver:

As explained at the beginning of the chapter, the FWave solver does not approximate well enough on the cells with smaller bathymetry due to the smaller water depth.

Due to calculations assuming the water height *h* is always positive, the FWave solver does not support wetting and drying scenarios. We work around this problem by assuming that when a wave moves from a wet region towards the shore; we presume an infinitely high wall reflecting the wave instead of wetting the dry cells.

HLLEM Solver

The HLLEM solver is an approximate Riemann solver used to compute numerical fluxes between cells in a computational domain. It is a modified version of the original HLL (Harten-Lax-van Leer) solver, specifically designed to improve accuracy and stability in various fluid dynamics simulations and is described in the literature [23, 6].

The core of the HLLEM solver is constructing the intermediate state U^* based on right and left states and the eigenvalues of the flux Jacobians. The intermediate state is then used to calculate the numerical flux ΔF between cells using the formula [23]:

$$\Delta F = \frac{S_R F_L - S_L F_R + S_L S_R (U_R - U_L)}{S_R - S_L} + \frac{S_L S_R}{S_R - S_L}(U_L - U_R)$$

We integrate the fluxes for the HLLEM solver into ExaHyPE 2 as detailed in Algorithm 4.

Accordingly, we implement the Riemann solver in Algorithm 5, which calculates the right- and left-going waves $A^{\pm \Delta Q}$ and we inherit the non-conservative products into the Riemann solver.

Algorithm 4 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
1: function FLUX($\mathbf{q}, n$)
2:     if $n = x$ then
3:         $\mathbf{F} \leftarrow \begin{pmatrix} hu \\ h \cdot u^2 \\ huv \\ 0.0 \end{pmatrix}$
4:     else
5:         $\mathbf{F} \leftarrow \begin{pmatrix} hv \\ huv \\ h \cdot v^2 \\ 0.0 \end{pmatrix}$
6:     end if
7:     return F
8: end function
```

Algorithm 5 HLLEM Riemann Solver

```
1: function RIEMANN($\mathbf{qL}, \mathbf{qR}, \mathbf{FL}, \mathbf{FR}, \lambda L, \lambda R$)
2:     $h^{Roe} \leftarrow \frac{1}{2}(h_l + h_r)$
3:     $\Delta\eta \leftarrow (h_r + b_r) - (h_l + b_r)$
4:     $\lambda_{max} \leftarrow \max(\lambda L, \lambda R)$
5:     $\Delta F \leftarrow \frac{FL}{2I} - \mathbf{1} \cdot \lambda_m ax \cdot \Delta\eta$
6:     $A^{-\Delta Q} \leftarrow \sum_{p:\lambda^{Roe}<0} \Delta F$
7:     $A^{+\Delta Q} \leftarrow \sum_{p:\lambda^{Roe}>0} \Delta F$
8:     if $n = x$ then
9:         $\Delta A^{+\Delta Q} \leftarrow A^{+\Delta Q} + \begin{pmatrix} 0 \\ \frac{gh^{Roe} \cdot \Delta\eta}{2} \\ 0 \\ 0 \end{pmatrix}$
10:        $\Delta A^{-\Delta Q} \leftarrow A^{+\Delta Q} - \begin{pmatrix} 0 \\ \frac{gh^{Roe} \cdot \Delta\eta}{2} \\ 0 \\ 0 \end{pmatrix}$
11:    else
12:        $\Delta A^{+\Delta Q} \leftarrow A^{+}\Delta Q + \begin{pmatrix} 0 \\ 0 \\ \frac{gh^{Roe} \cdot \Delta\eta}{2} \\ 0 \end{pmatrix}$
13:        $\Delta A^{-\Delta Q} \leftarrow A^{+\Delta Q} - \begin{pmatrix} 0 \\ 0 \\ \frac{gh^{Roe} \cdot \Delta\eta}{2} \\ 0 \end{pmatrix}$
14:    end if
15:    return $\lambda_{max}$
16: end function
```

Advantages of the HLLEM Solver:

The HLLEM solver enhances accuracy in regions near shorelines and effectively handles wetting and drying scenarios. It is particularly well-suited for wetting and drying scenarios, where it maintains a small volume of water in dry cells to prevent numerical instabilities.

4.2. ADER-DG

Arbitrary Derivative Discontinuous Galerkin (ADER-DG) is a numerical method employed to solve hyperbolic partial differential equations (PDEs) such as shallow water equations. The combination of ADER and DG provides arbitrary high-order accuracy in space and time. The DG method's ability to handle discontinuities makes it suitable for complex geometries and problems with sharp gradients.

The Discontinuous Galerkin (DG) methods represent both the source terms and the fluxes within cells by a (high-order) polynomial [18], unlike the finite volume methods, which allow discontinuities between cells, making it more suitable for problems with complex geometries.

The DG method approximates the state vector Q within each cell with [10]:

$$Q(x,t) = \sum_{l=1}^{N} Q^l(t)\phi_l(x)$$

where, $\phi_l(x)$ is typically a polynomial with l degrees of freedom.

On the other hand, the single-step ADER approach reduces the computational overhead compared to traditional multi-step time integration methods by integrating both space and time derivatives in a unified framework.

For the integration in time, the weak form of the PDE is used as suggested in paper [17]:

$$\int_{t_n}^{t_{n+1}} \int_T \Phi^T(Q(x,t) + \nabla F(Q) - S(Q))dxydt = 0$$

For a detailed introduction to ADER-DG methods, the original paper [5] should be referred to.

In ExaHyPE 2, the ADER-DG approach is defined for SWEs by implementing the flux calculation using the flux function as detailed in Algorithm 4, the largest eigenvalue as described in Algorithm 7 to determine the time step size and non-conservative products (NCPs) using the algorithm described in the Algorithm 6. The generic Riemann solver in ExaHyPE 2 is used to solve the Riemann problem.

Algorithm 6 Non-Conservative Product

Input: $\mathbf{q}$: state vector with elements h, hu, hv, b
$\nabla\mathbf{q}$: gradient vector with elements $h_{\text{grad}}, hu_{\text{grad}}, hv_{\text{grad}}, b_{\text{grad}}$
$n \in \{x, y\}$: normal direction
Output: **NCP**: array containing non-conservative product in normal direction

```
1: function NCP($\mathbf{q}, \nabla\mathbf{q}, n$)
2:     if $n = x$ then
3:         NCP ← $\begin{pmatrix} 0.0 \\ gh(b_{\text{gradQ}} + h_{\text{gradQ}}) \\ 0.0 \\ 0.0 \end{pmatrix}$
4:     else
5:         NCP ← $\begin{pmatrix} 0.0 \\ 0.0 \\ gh(b_{\text{gradQ}} + h_{\text{gradQ}}) \\ 0.0 \end{pmatrix}$
6:     end if
7:     return NCP
8: end function
```

Algorithm 7 Eigenvalues

Input: $\mathbf{q}$: state vector with elements h, hu, hv, b $\quad n \in \{x, y\}$: normal direction
Output: **NCP**: the maximal eigenvalue on that timestep for calculating the timestep

```
1: function MAXEIGENVALUE($\mathbf{q}, n$)
2:     if $n = x$ then
3:         $u \leftarrow u$
4:     else
5:         $u \leftarrow v$
6:     end if
7:     return $max(u + gh, u - gh)$
8: end function
```

Advantages of ADER-DG Schemes:

The ADER-DG method achieves arbitrary high-order accuracy in both spatial and temporal discretizations by representing the cells by polynomials and can handle discontinuous state vectors.

The ADER approach's single-step time integration reduces computational overhead compared to traditional multi-step methods. This efficiency makes the ADER-DG scheme well-suited for large-scale simulations where computational resources and time are critical.

4.3. Limiting

The unlimited ADER-DG algorithm is prone to numerical oscillations (Gibbs phenomenon) in the presence of steep gradients or shock waves [18]. These oscillations can lead to instabilities in cells where the ocean depth is shallow. ExaHyPE 2 offers limiting solvers that partition the domain into two discrete regions and utilize two different solvers on these domains to address this. This thesis uses the ADER-DG solver for most of the domain while switching to the HLLEM FV solver on the problematic cells where the water depth is less than 10 meters, thus resulting in wetting-drying scenarios near shorelines. Figure 4.1 shows the distribution of cells between two solvers.

4.4. Particles

In ExaHyPE 2, particles track specific points within the computational domain, serving as tracers for detailed analysis.

In this thesis, we implement two tracers representing DART buoys near the Japanese coast. The location of the DART buoys was obtained from the National Data Buoy Center (NDBC) [15]. The data recorded by these tracers are then compared with the historical data from the 2011 Tōhoku tsunami and the solver from the research [22] to evaluate the accuracy of the solvers.

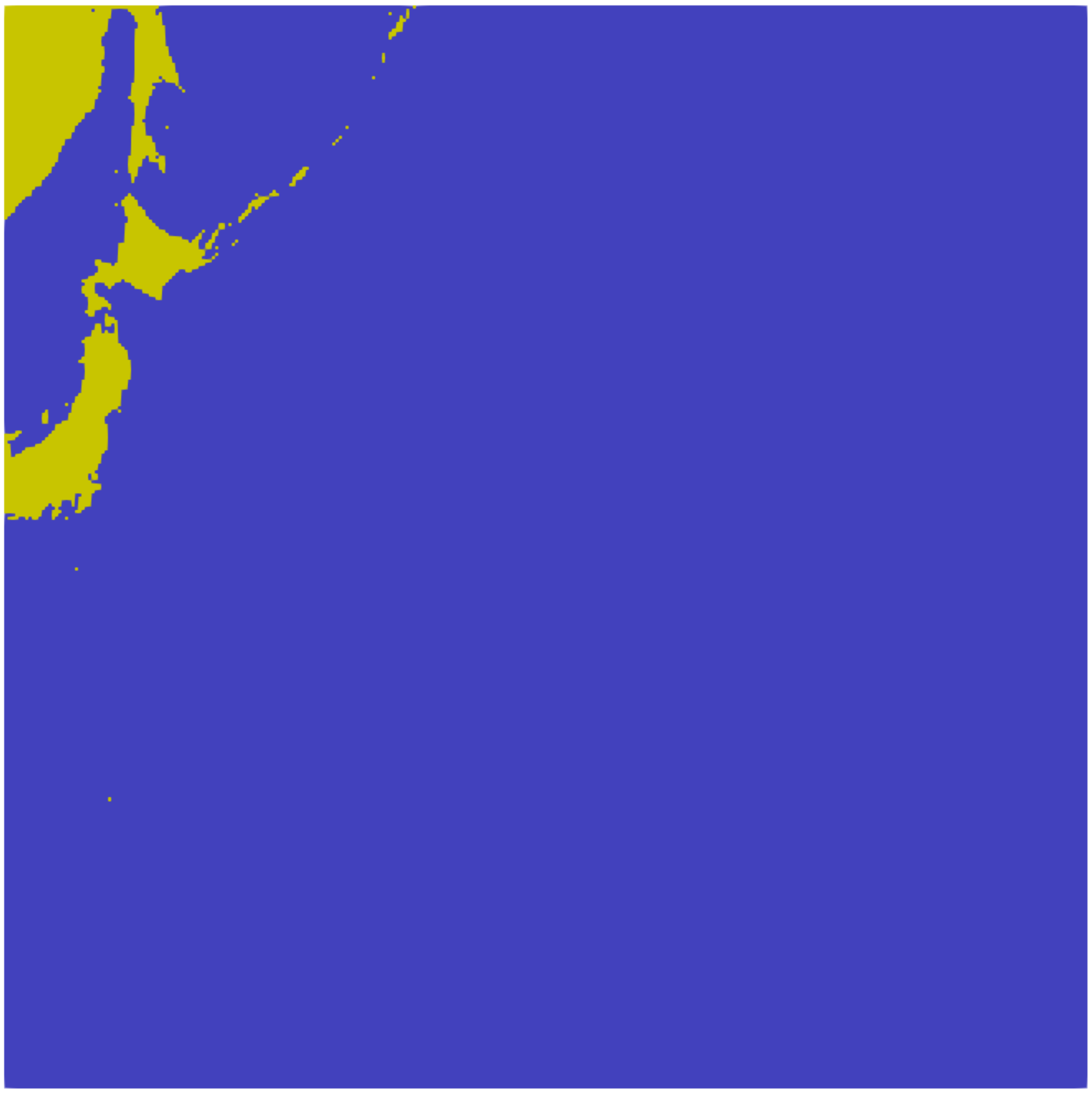


Figure 4.1.: The split of the cells for the limiting solver with min depth 4. For the yellow domain HLLEM FV solver is used, while for the blue domain the ADER-DG solver is used.

5. UM-Bridge in ExaHyPE 2

5.1. UM-Bridge

UM-Bridge (**u**ncertainty quantification and **m**odeling **bridge**) provides an interface between the uncertainty quantification methods and the models [20]. One of the most common applications of uncertainty quantification is quantifying the tsunami source location [21]. In other areas, it can be used to quantify the material parameters of the elastic simulation [24].

In most cases, the UQ algorithms treat the forward model as a function $y = f(x)$, abstracting the implementation details of the simulation model, thus not requiring any knowledge about $f(x)$ other than its results [20]. UQ is designed to help this abstraction by physically decoupling the forward model and the uncertainty quantification. It allows the same UQ algorithms and the forward simulations to be developed in different languages and even the same UQ algorithm to be used on multiple fields and simulations.

The general client-server model for a tsunami simulation might look like Figure 5.1, where the application handles all the communication between the server and the UQ model via an HTTP connection, thus allowing them to run on even different environments. For example, the computationally heavy simulation can be run on a supercomputer powered by thousands of processor cores. In contrast, the computationally less intensive statistical UQ model can be run on a computer with much less computational power.

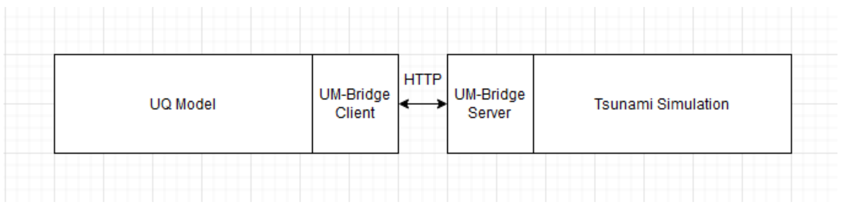


Figure 5.1.: The UM-Bridge interface

In this thesis, we plan to focus on the tsunami simulations and quantifying the source location of the tsunami displacement on the Tōhoku tsunami using the solvers we already have implemented in Chapter 4. Therefore, we implement a client-server UM-Bridge interface to be able to implement the UQ algorithms.

Server

The server is a forward model that solves the forward problem $y = f(x)$. It takes the proposed displacement (x, y) for the tsunami as an input. It returns the wave's arrival time and maximum wave height $y = (t_0, h_0, t_1, h_1)$ during the simulation in specific probe points implemented in Chapter 4.4. The forward model is implemented in the same fashion as in [21].

The server essentially has three functions. Where *get_input_sizes()* returns the number of dimensions in the simulation and *get_output_sizes()* returns the size of the y tuple. On the other hand, a *call(x)* method takes the position of the displacement for the tsunami simulation and returns the tuple y to the client.

The server is typically run in the local host for testing purposes. We define the server's port using UM-Bridge via the following:

```
model = TsunamiModel()
umbridge.serve_models([model], 4242)
```

Client

The client is where the UQ algorithms are implemented. In this thesis, we plan to implement a Bayesian algorithm to solve the inverse problem and quantify the source location of the given tsunami. Many evaluation-based statistical algorithms can be used for this purpose. We start by implementing the Metropolis Hastings Markov Chain Monte Carlo algorithm in Chapter 5.2.3 then extending it to multiple layers using the Multilevel Markov Chain Monte Carlo algorithm as proposed in Chapter 5.2.4.

The client is first informed on where the server lies using UM-Bridge:

```
print(umbridge.supported_models("http://0.0.0.0:4242"))
model = umbridge.HTTPModel("http://0.0.0.0:4242", "forward")
```

The client first queries the server about the input and output specifications and then calls the *call()* function from the server via an HTTP connection. It runs the user's UQ algorithm and uses the server as a black box implementation for the $y = f(x)$ function.

Figure 5.2 illustrates the communication between various components of the UM-Bridge implementation of a UQ algorithm.

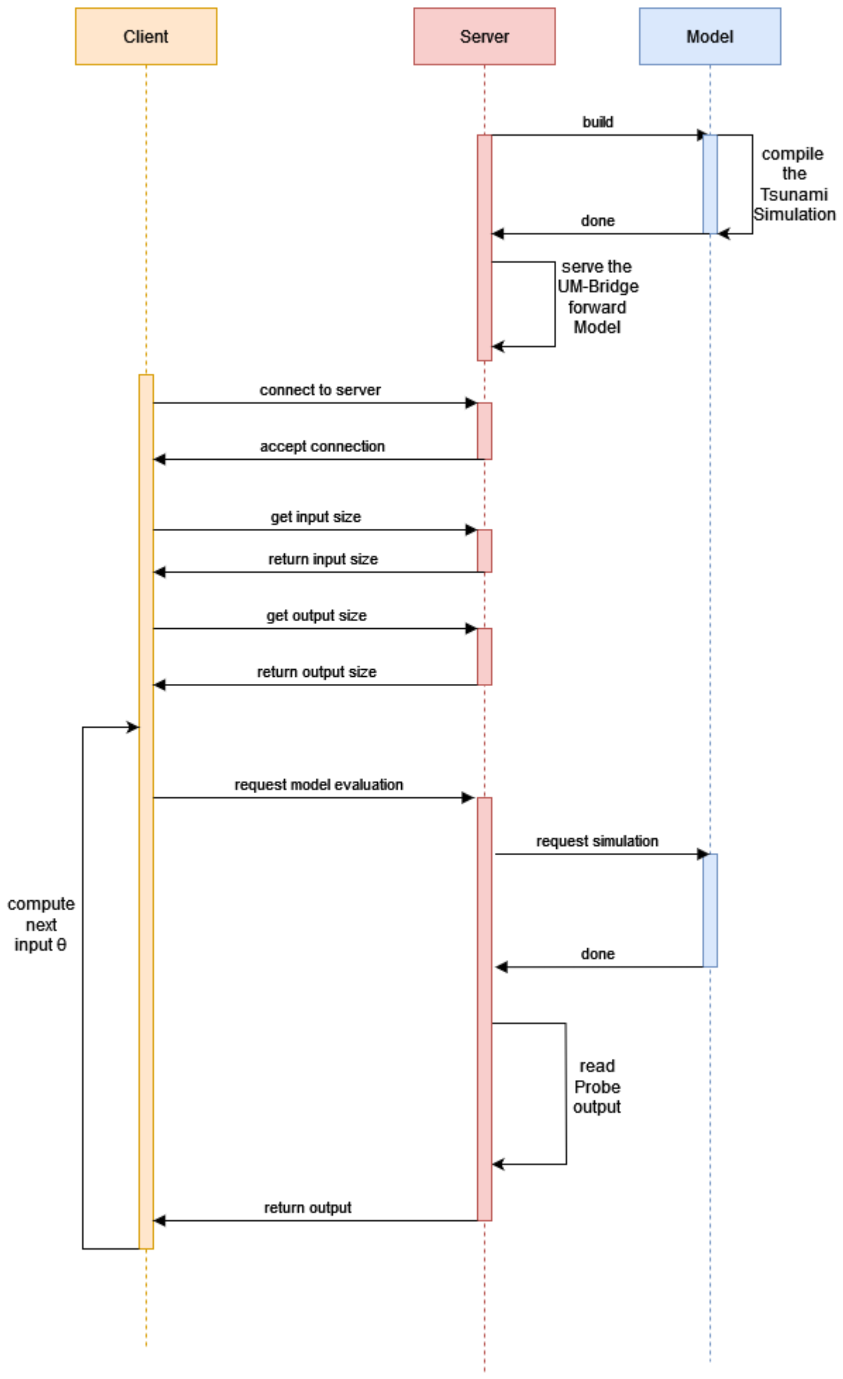


Figure 5.2.: The UM-Bridge sequence diagram

5.2. Monte Carlo Algorithms

5.2.1. Monte Carlo

We will start by defining a simple Monte Carlo method over the tsunami displacement. Monte Carlo methods rely on repeated random sampling to guess random parameters or to access the general functions of those parameters [11].

5.2.2. Markov Chain Monte Carlo

The Markov Chain Monte Carlo (MCMC) method is a powerful and simple tool for solving Bayesian inverse problems. The MCMC method uses the Bayesian probability theory to calculate a likelihood function and a posteriori distribution for the unknown parameter θ [1].

5.2.3. Metropolis Hastings Markov Chain Monte Carlo

The Metropolis-Hastings (MH) Markov Chain Monte Carlo (MCMC) algorithm is a method for sampling from a probability distribution when direct sampling is difficult.

We begin with an initial guess θ^0, which represents the initial displacement for the tsunami. This value is chosen arbitrarily since it is the start of the chain.

We generate a new proposal θ' for each iteration based on the current state θ^i. This generation step works like a random walk in a random direction as the new proposition can be generated using a normal distribution centered at the current state.

After the proposal, we calculate how likely we are to accept this new state as our new θ. The acceptance probability $\alpha(\theta'|\theta^i)$ is determined by how well the θ' fits the target distribution compared to the current θ^i, calculated using our likelihood function. If θ' gives a better fit, we accept the θ' as the new state for our Markov Chain θ^{i+1}, else we might accept it with a lower probability depending on the likelihood change. This prevents the function from being stuck on a local minimum state.

This process repeats for many iterations. Over time, the parameters θ^i will converge to the target distribution.

The main advantage of this method is its availability since this algorithm uses no information specific to the model. Instead, it only uses the results from the direct evaluations of the forward model to calculate the likelihood function. Thus, no particular information for the model as derivatives or fluxes needs to be passed to the client [22], making this method more universal for other forward models. However, it requires many iterations for the forward model to be calculated to get a result, and, therefore, it can be very computationally intensive. In Section 5.2.4, we aim to present an algorithm to solve this issue by introducing a layered approach to the algorithm.

Algorithm 8 Metropolis Hastings MCMC

```
Require: Markov Chain $\{\theta^i\}_{i=0}^N$
 1: Choose starting parameter $\theta^0 \in \mathbb{R}^m$
 2: for $i = 0, \dots, N-1$ do
 3:     Draw proposal $\theta'$ from proposal distribution $q(\theta'|\theta^i)$
 4:     Compute acceptance probability
            $$\alpha(\theta'|\theta^i) = \min\left(1, \frac{v(\theta')q(\theta^i|\theta')}{v(\theta^i)q(\theta'|\theta^i)}\right)$$
 5:     Draw a random number $r \in [0,1]$
 6:     if $r < \alpha(\theta'|\theta^i)$ then
 7:         Accept proposal: $\theta^{i+1} = \theta'$
 8:     else
 9:         Reject proposal: $\theta^{i+1} = \theta^i$
10:     end if
11: end for
```

5.2.4. Multilevel Markov Chain Monte Carlo

To get around the previous model's computational intensity, the Multilevel (ML) Markov Chain Monte Carlo (MCMC) defines a hierarchy of models, ranging from cheap to compute rough approximation to the most accurate full model.

We start with Markov Chains θ_l^i for each level and run conventional MCMC as described in Section 5.2.3. For any other level for each sample j at level l we draw a combined proposal $\theta_l'^j = (\theta_{l,C}'^j, \theta_{l,F}'^j)$ where $\theta_{l,C}'^j$ is drawn from a level $l-1$ chain with subsampling rate ρ_l and $\theta_{l,F}'^j$ is drawn from a proposal density $q_l(\theta_{l,F}'^j|\theta_{l,F}^j)$.

Then the acceptance probability $\alpha(\theta_{l,C}'^j, \theta_{l,F}'^j)$ is defined as:

$$\alpha(\theta_l'^j, \theta_l^j) = \min\left\{1, \frac{v_l(\theta_l'^j)q_l(\theta_{l,F}^j|\theta_{l,F}'^j)}{v_l(\theta_l^j)q_l(\theta_{l,F}'^j|\theta_{l,F}^j)} \frac{v_{l-1}(\theta_{l,C}^j)}{v_{l-1}(\theta_{l,C}'^j)}\right\}$$

Then, the proposal is accepted or rejected using the same logic as Section 5.2.3 by drawing a random number between 0 and 1. The specific algorithm used for this method is taken from [22] and given as Algorithm 9.

Algorithm 9 Multilevel MCMC

```
Require: Markov Chains $\{\theta_l^i\}_{i=0}^{N_l}$ for all levels $l \in \{0, \dots, L\}$. On level 0, run a conventional MCMC, delivering samples $\theta_0^i$.
  for level $l = 1, \dots, L-1$ do
    Choose starting point $\theta_l^0$ with coarse component from next coarser starting point $\theta_{l-1}^0$
    for sample $j = 1, \dots, N_l$ do
      Given $\theta_l^j$, generate proposal $\theta_l'^j = \begin{cases} \theta_{l,C}'^j \\ \theta_{l,F}'^j \end{cases}$ where
        • $\theta_{l,C}'^j$ is drawn from a level $l-1$ chain with subsampling rate $\rho_l$ and
        • $\theta_{l,F}'^j$ is drawn from a proposal density $q_l(\theta_{l,F}'^j | \theta_{l,F}^j)$.
      Compute acceptance probability
        $$\alpha(\theta_l'^j, \theta_l^j) = \min\left\{1, \frac{v_l(\theta_l'^j) q_l(\theta_{l,F}^j | \theta_{l,F}'^j)}{v_l(\theta_l^j) q_l(\theta_{l,F}'^j | \theta_{l,F}^j)} \frac{v_{l-1}(\theta_{l,C}^j)}{v_{l-1}(\theta_{l,C}'^j)}\right\}$$
      Draw a random number $r \in [0, 1]$
      if $r < \alpha(\theta_l'^j, \theta_l^j)$ then
        Accept proposal: $\theta_l^{j+1} = \theta_l'^j$
      else
        Reject proposal: $\theta_l^{j+1} = \theta_l^j$
      end if
    end for
  end for
```

6. Results

6.1. Evaluation of the Solvers

This chapter shows the simulation results using the various solvers implemented in Chapter 4 to validate the consistency of solvers. The results are displayed using 2D-surface plots and line plots. We have chosen the radial dam break scenario and the 2011 Tōhoku tsunami to test out solvers.

6.1.1. Radial Dam Break Scenario

The radial dam break scenario is a classic test case in models that are used for fluid dynamics. It is particularly useful for validating the accuracy and stability of the solvers with well-defined initial conditions.

In this scenario, the simulation domain is a square domain with length N. The height, bathymetry and velocity distribution is given by:

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

$$hu(x,y) = 0.0\,\mathrm{m}^2/\mathrm{s},$$

$$hv(x,y) = 0.0\,\mathrm{m}^2/\mathrm{s},$$

$$b(x,y) = -1.0\,\mathrm{m}.$$

Here, the water height is slightly larger at the center of the domain with a radius of $N/10$ meters, representing the initial "dam" of the water. Which then propagates outwards to the rest of the domain.

For this simulation, we chose $N = 10$ meters. All methods are simulated using adaptive time-stepping using the TSR factor of 0.4 and the mesh depth of 5 levels.

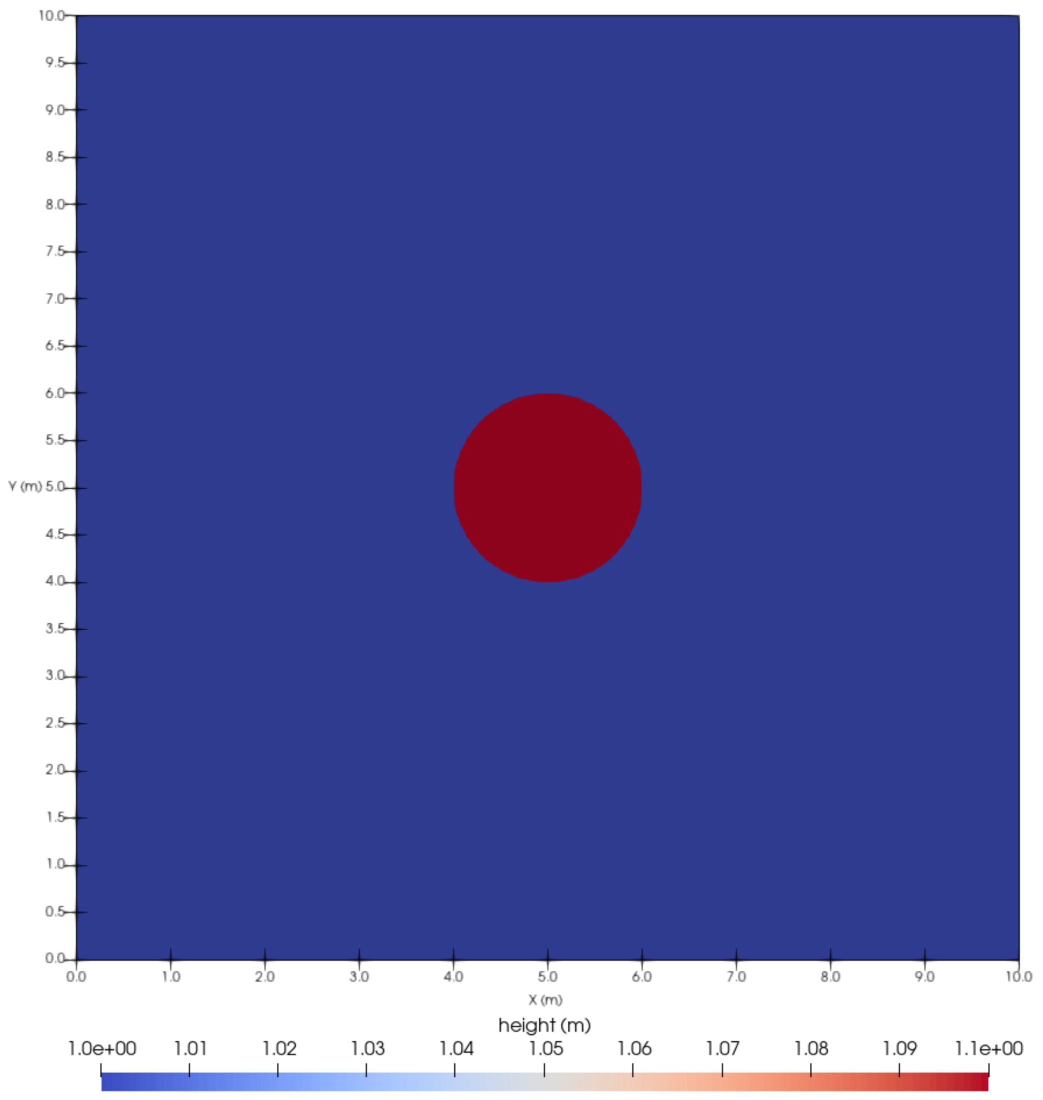
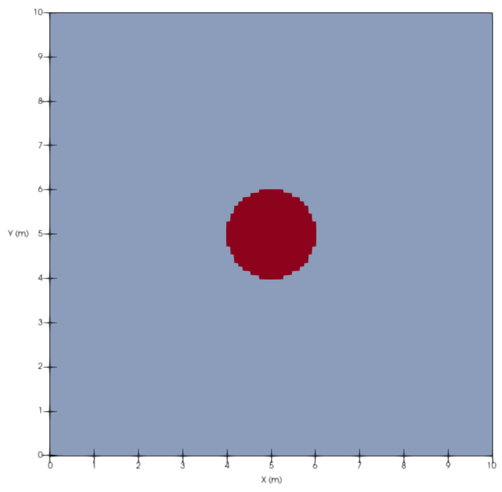


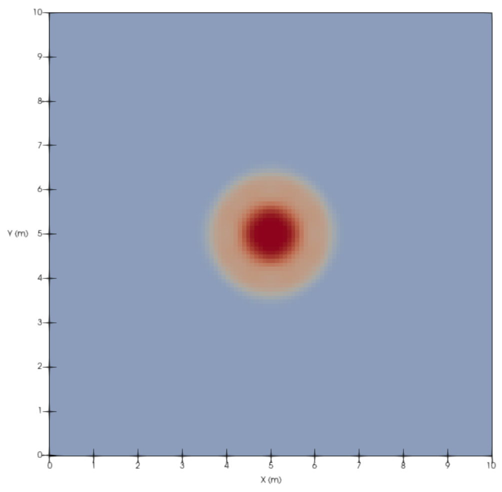
Figure 6.1.: Initial conditions for the dam break scenario

In the following, we show the results for all solvers that are implemented in the Chapter 4 given the same scenario on a 2D-surface plot over 1 second, as well as the comparative results for all chapters plotting their height and momentum shifts over time using line plots.

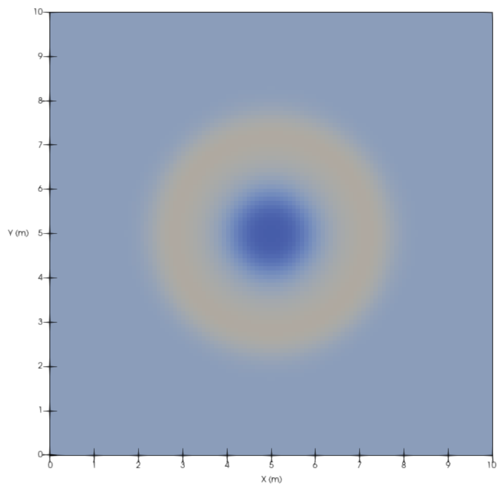
Generic Riemann Solver



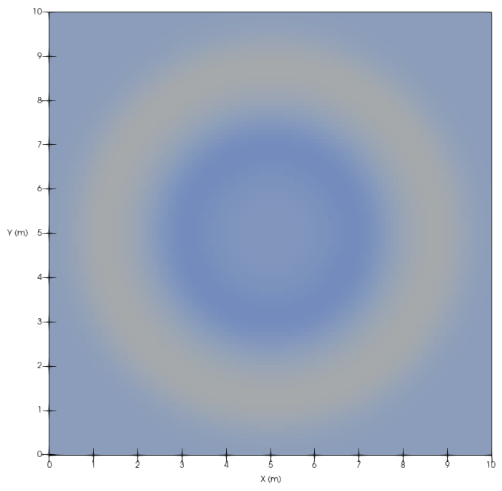
(a) t = 0.0 s



(b) t = 0.1 s



(c) t = 0.5 s



(d) t = 1.0 s

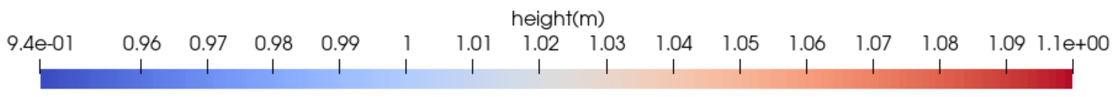
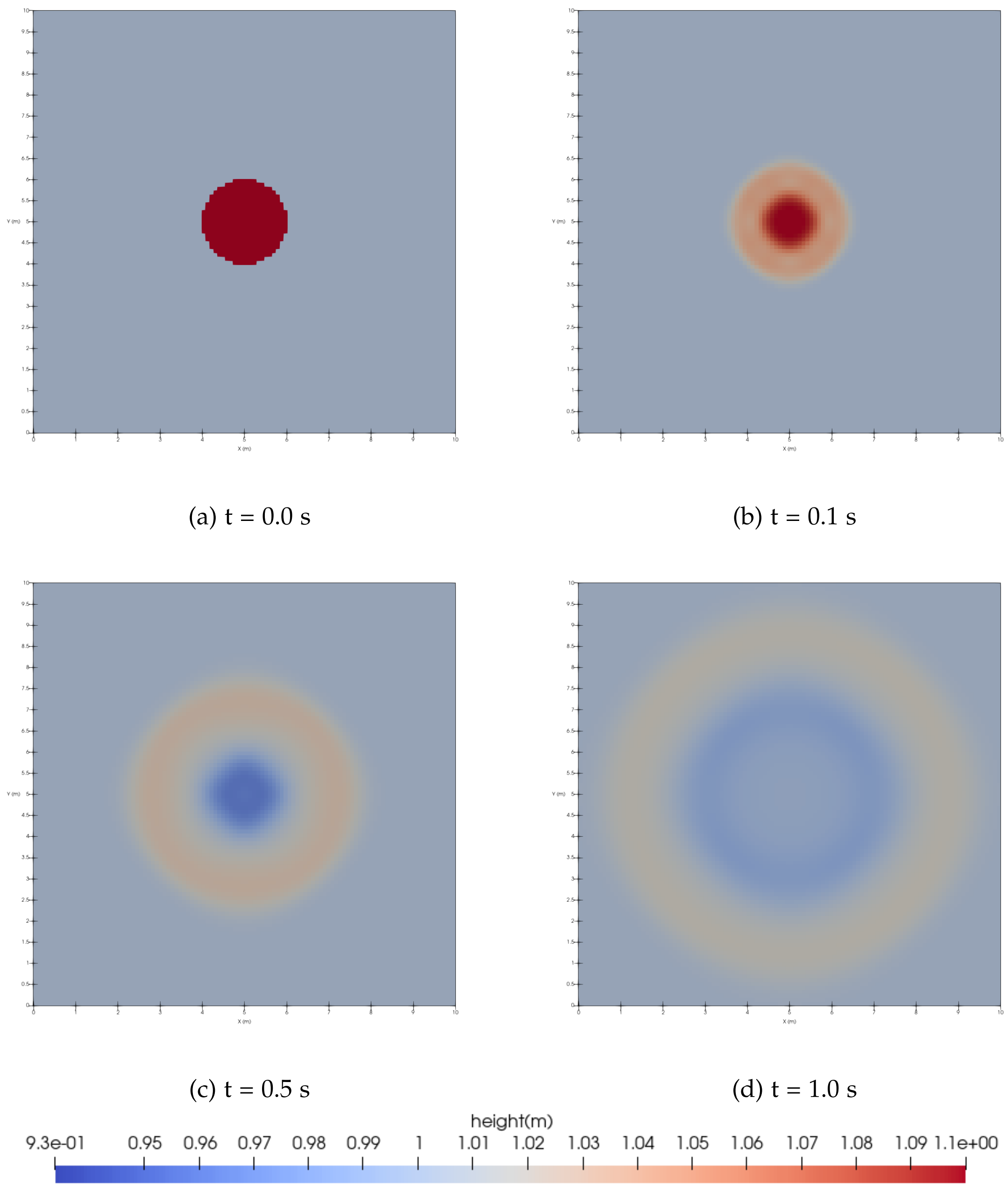


Figure 6.2.: 2D-surface plot over time for dam break scenario with the generic Riemann solver.

FWave Solver



(a) t = 0.0 s

(b) t = 0.1 s

(c) t = 0.5 s

(d) t = 1.0 s

Figure 6.3.: 2D-surface plot over time for dam break scenario with the FWave solver.

HLLEM Solver

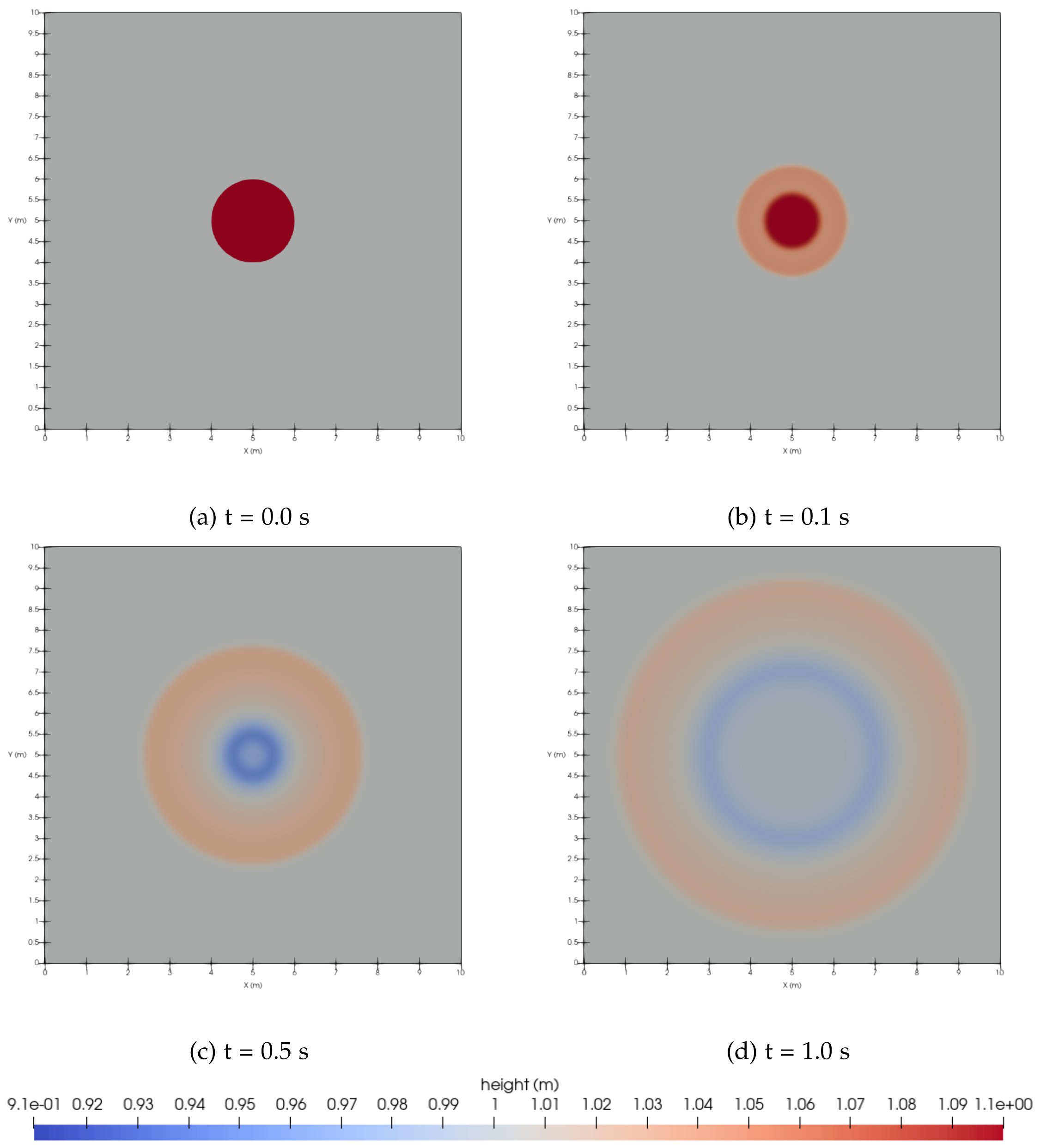


Figure 6.4.: 2D-surface plot over time for dam break scenario with the HLLEM solver.

ADER-DG Solver

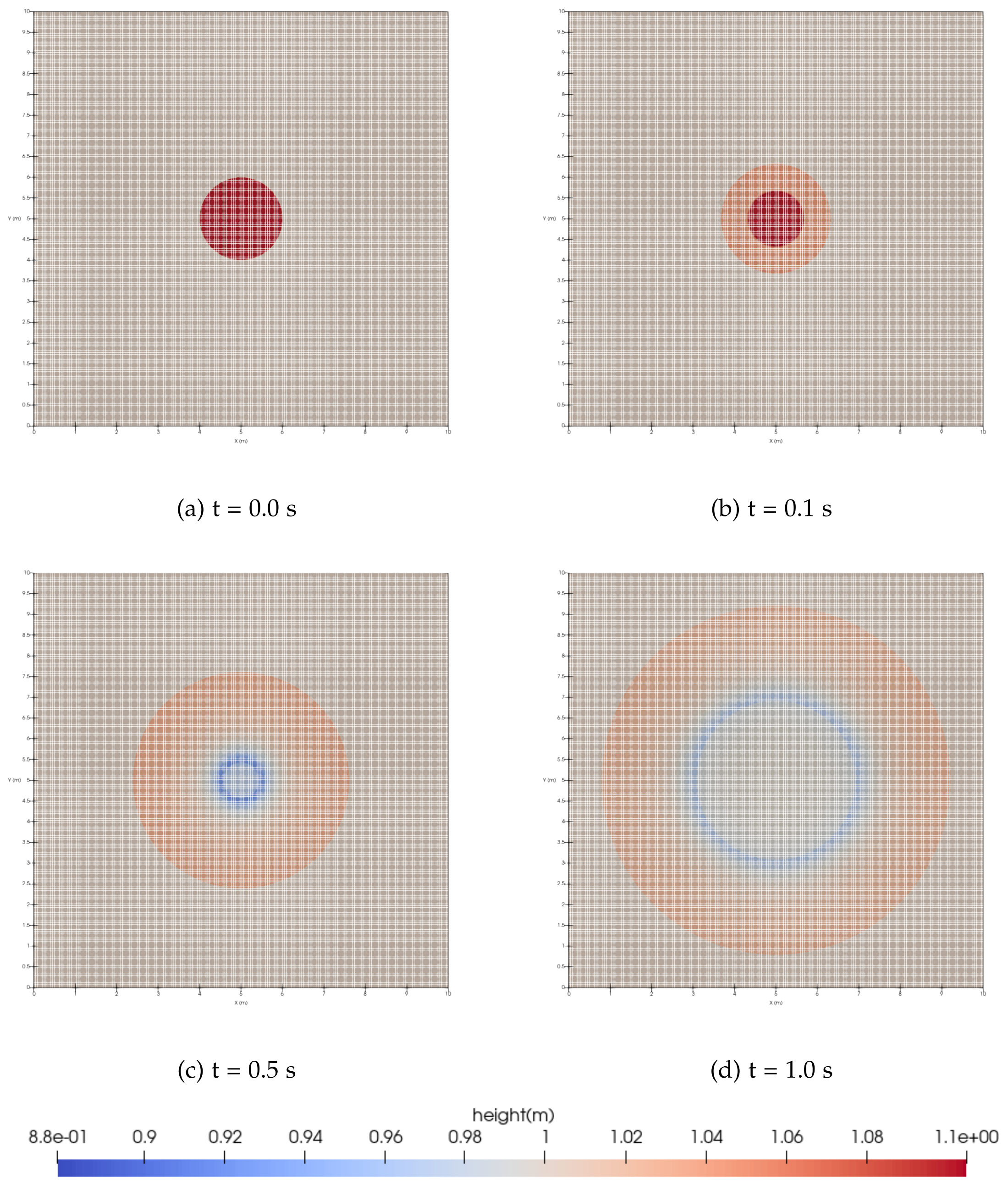
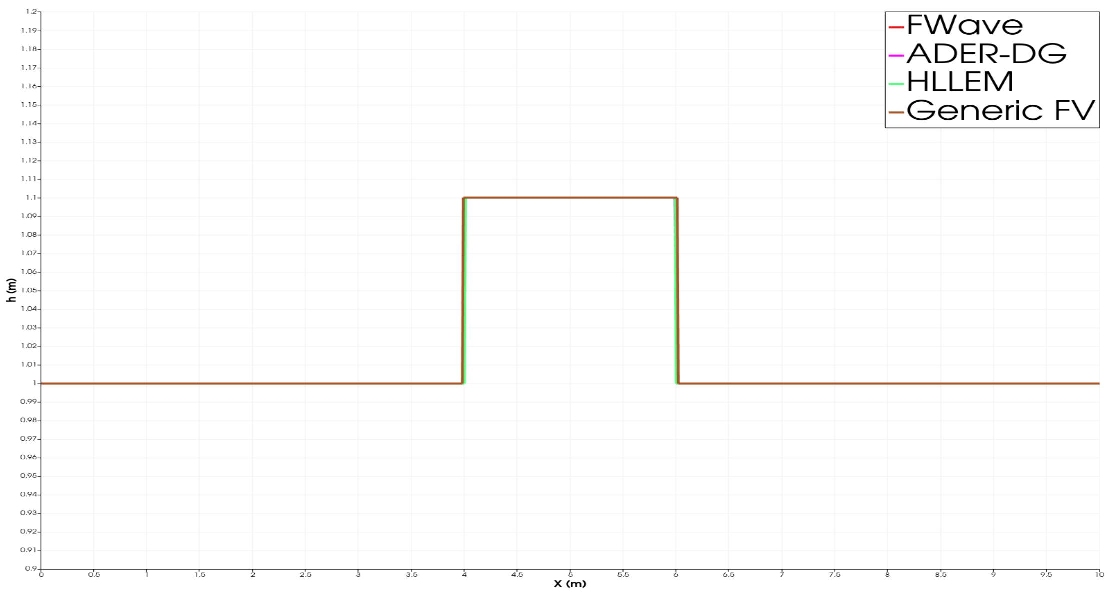
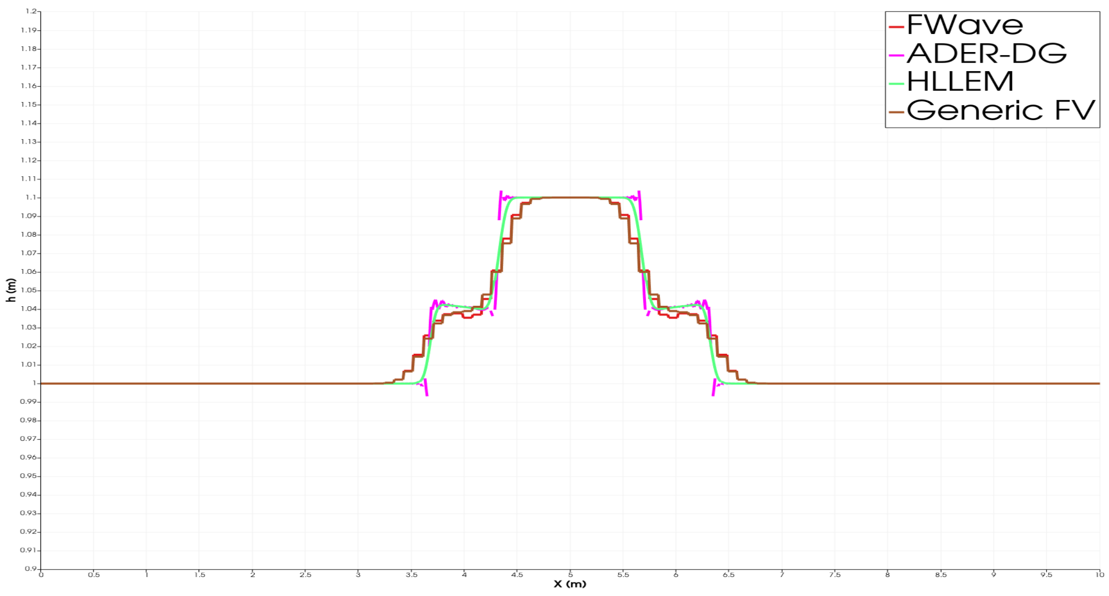


Figure 6.5.: 2D-surface plot over time for dam break scenario with the ADER-DG solver.

Comparison

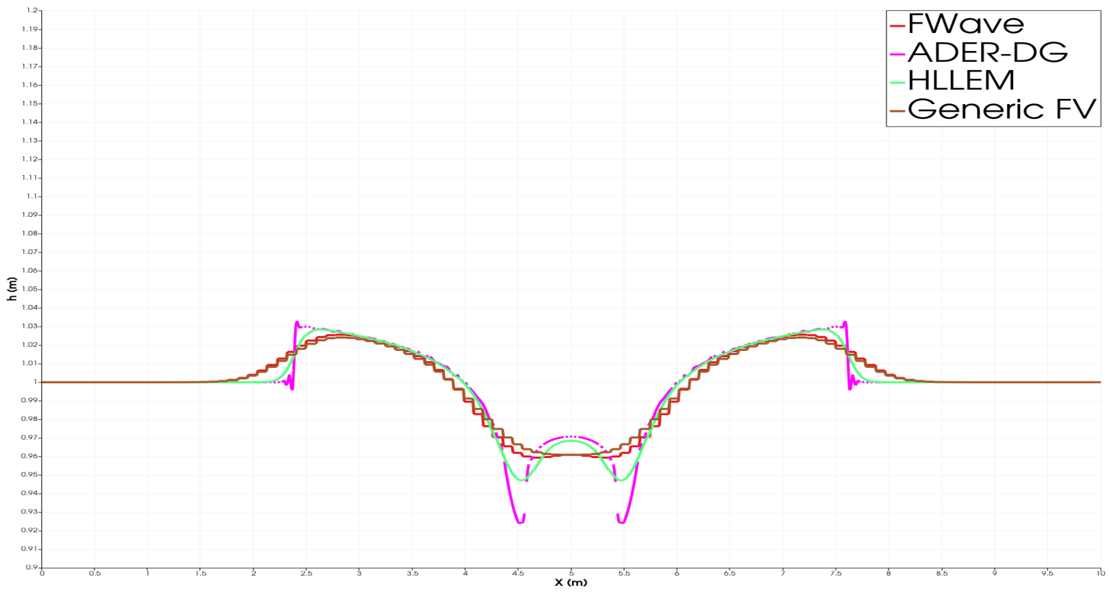


(a) t = 0.0 s

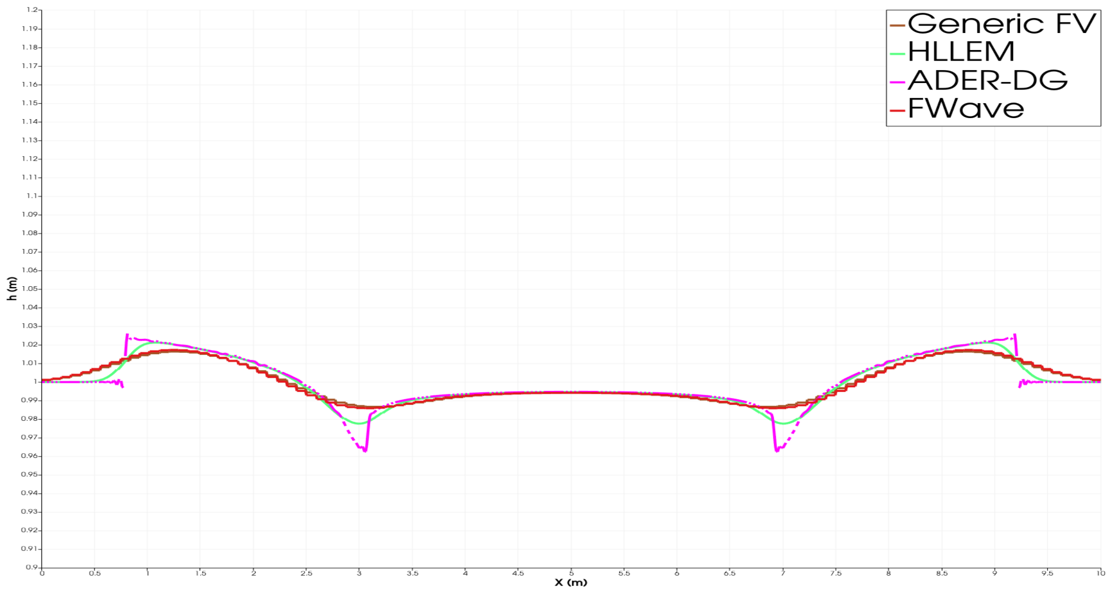


(b) t = 0.1 s

Figure 6.6.: Line plot of the height along the x-axis comparing results from different solvers in timesteps $t = 0\,\mathrm{s}$ and $t = 0.1\,\mathrm{s}$.

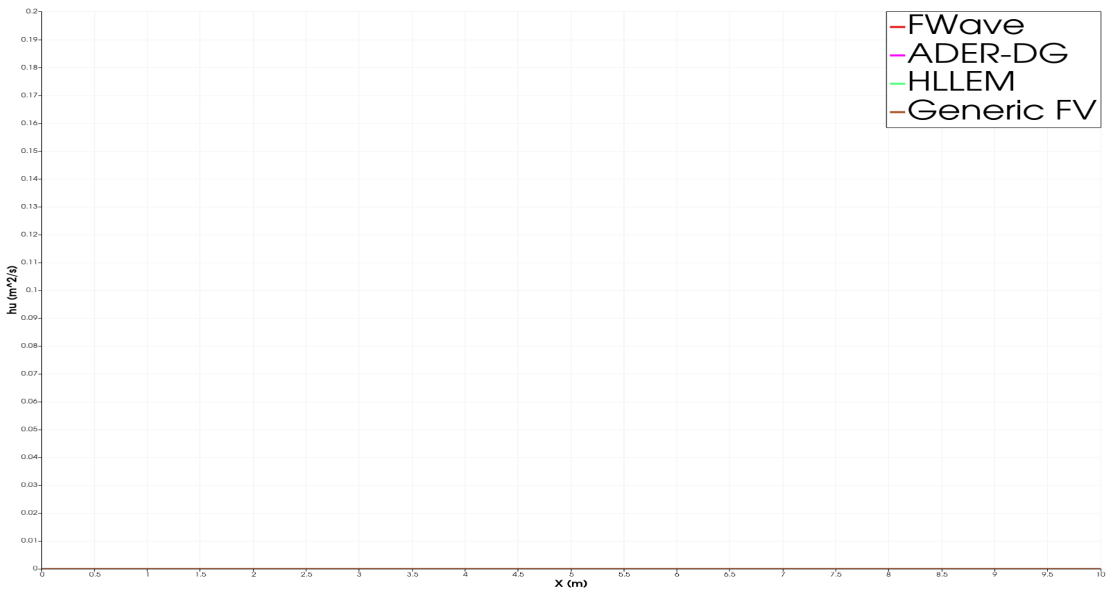


(a) t = 0.5 s

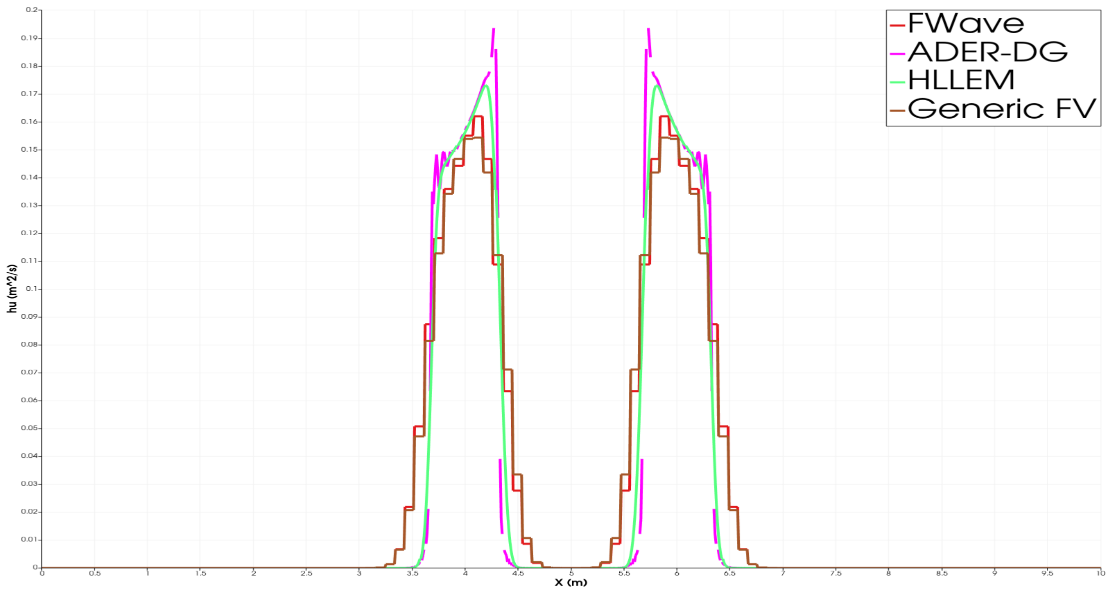


(b) t = 1.0 s

Figure 6.7.: Line plot of the height along the x-axis comparing results from different solvers in timesteps $t = 0.5\,\text{s}$ and $t = 1.0\,\text{s}$.

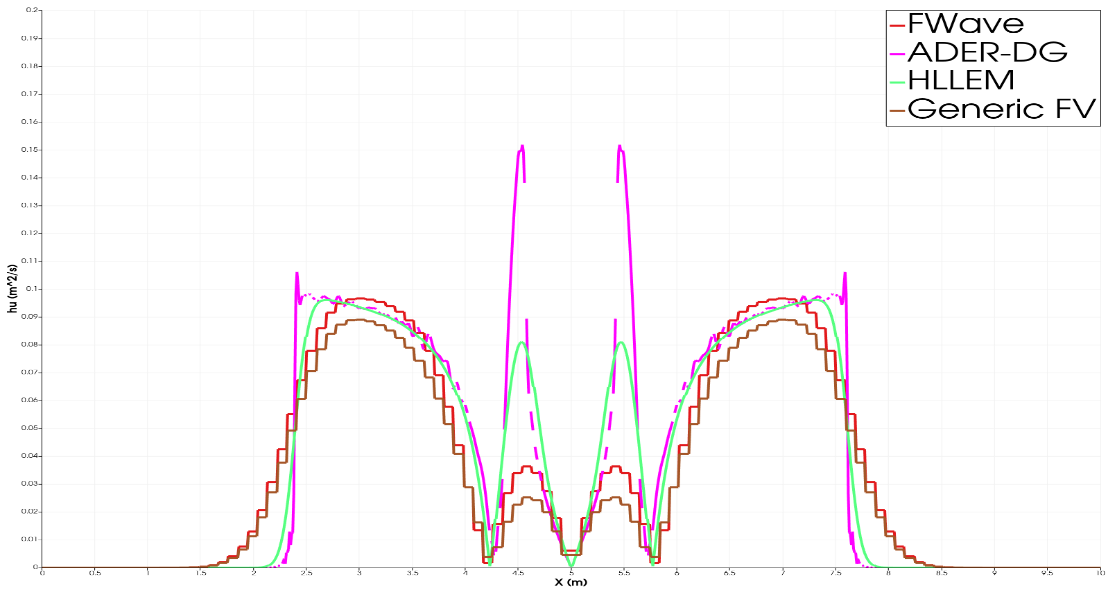


(a) t = 0.0 s

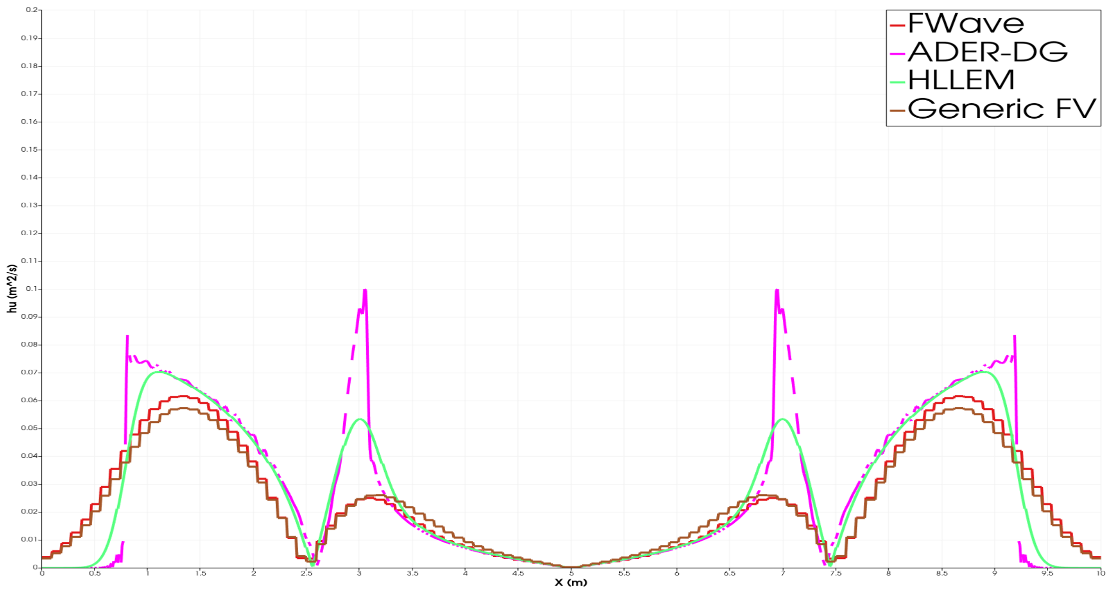


(b) t = 0.1 s

Figure 6.8.: Line plot of the absolute value of momentum in the x-direction along the x-axis comparing results from different solvers in timesteps $t = 0\,\text{s}$ and $t = 0.1\,\text{s}$.



(a) t = 0.5 s



(b) t = 1.0 s

Figure 6.9.: Line plot of the absolute value of momentum in the x-direction along the x-axis comparing results from different solvers in timesteps $t = 0.5\,\mathrm{s}$ and $t = 1.0\,\mathrm{s}$.

From Figures 6.3 to 6.5, we observe the behavior of the water height over the domain with different solvers. In all cases, the solvers show symmetrical wave propagation originating from the center of the domain. As the water in the center collapses, it generates circular waves that move towards the domain's boundaries. Notably, the total water mass within the domains is preserved across all solvers, which shows that the solvers respect the mass conservation laws.

Figures 6.6 to 6.9 show a side-to-side comparison of different solvers through line plots that track the water height and momentum, respectively. The results show that all four solvers have similar wavefront propagation speeds throughout 1 second, which reflects the rate at which the disturbances travel through the domain. Throughout the simulations, all solvers reach a maximum wave speed of $\lambda \approx 0.16\,\mathrm{m/s}$.

However, the solvers disagree on the height of the shock wave produced after timestep $t = 0.5\,\mathrm{s}$; the HLLEM and ADER-DG solvers produce a more pronounced shock wave than the FWave and generic Riemann solvers. This is likely due to the latter two solvers' handling of the shallow water heights. Another notable difference we can observe is the ADER-DG solver having significant jumps on height and momentum representations, which are caused by approximating the cells with polynomials while having a discontinuous initial condition.

6.1.2. Tōhoku Tsunami

We implement a real-world tsunami event to further test the accuracy of solvers implemented so far. The 2011 Tōhoku tsunami was chosen for this purpose. The tsunami was simulated using adaptive time-stepping with a TSR factor 0.4 and a mesh depth of 5. The initial conditions for the tsunami can be found in Figure 6.10.

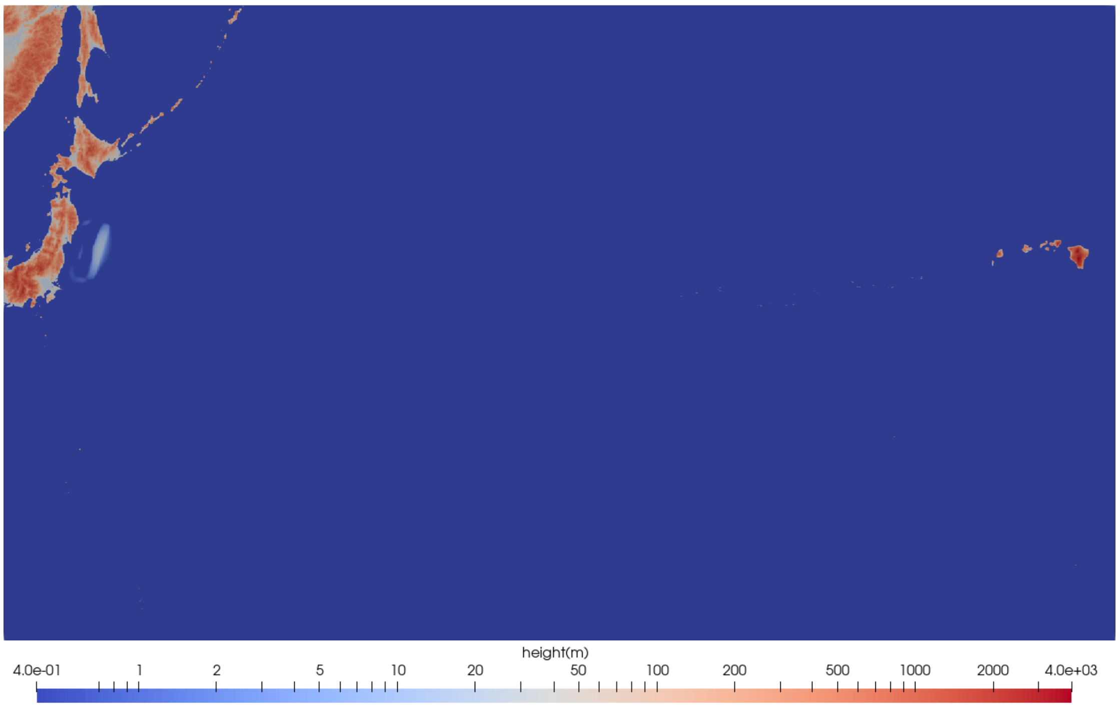


Figure 6.10.: Initial conditions of Tōhoku tsunami displaying $h + b$ in logarithmic scale.

FWave

Our initial simulation of the Tōhoku tsunami is conducted using an FWave solver. A maximum wave speed of $\sqrt{gh} = 310\,\mathrm{m/s}$ is reached during the simulation, which aligns with theoretical expectations for such events. Even though the results are promising, the simulation showed that the tsunami waves reached the Soma shoreline at approximately $t \approx 2160\,\mathrm{s}$. This result is inconsistent with the real-world observations, where the tsunami arrived at the Soma shore around 9 minutes after the earthquake, or $t \approx 540\,\mathrm{s}$.

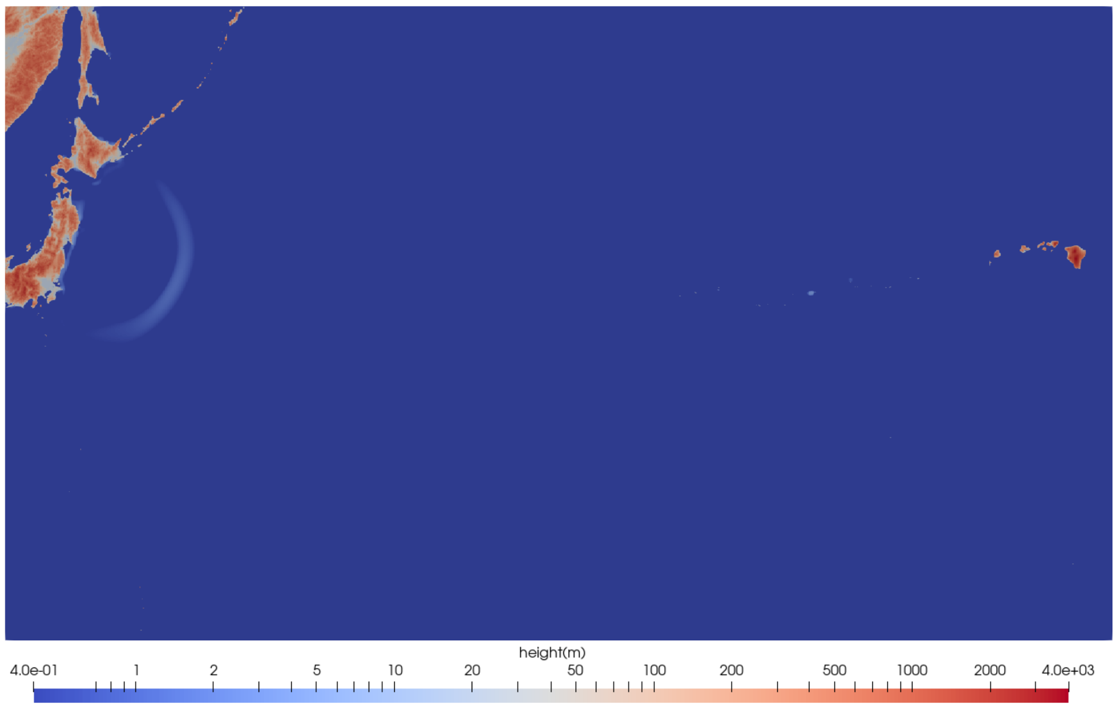


Figure 6.11.: Tōhoku tsunami approaching Soma at $t = 1607\,\mathrm{s}$ displaying $h + b$ in logarithmic scale using the FWave solver with TSR factor of 0.4.

The simulation produces artifacts after the timestep $t = 3000\,\mathrm{s}$, as shown in Figure 6.12. These artifacts become more violent over time, increasing the maximum wave speed and making the simulation unstable, which is fixed by improving the time-stepping in the following section.

To further investigate the FWave solver's performance, we compared the simulated wave heights and arrival times at two available DART buoys, 21418 and 21419, from the National Data Buoy Center (NDBC) near the Japanese coast with real-world data. Additionally, we compared the results of our solver to the outputs from the ExaHyPE 1 simulation, which are reported to agree with the real-world data in the literature [22]. This comparison was made using the probes implemented in Chapter 4.4:

- **Buoy 1:**
  - Real-world data: $t_0 \approx 30\,\mathrm{min} = 1800\,\mathrm{s}$, $h_0 \approx 2\,\mathrm{m}$
  - ExaHyPE 1 output: $t_0 = 1813.8\,\mathrm{s}$, $h_0 = 1.85232\,\mathrm{m}$
  - Solver output: $t_0 = 1777.952\,\mathrm{s}$, $h_0 = 1.58\,\mathrm{m}$

- **Buoy 2:**
  - Real-world data: $t_1 \approx 90\,\text{min} = 5400\,\text{s}$, $h_1 \approx 0.7\,\text{m}$
  - ExaHyPE 1 output: $t_1 = 5278.8\,\text{s}$, $h_1 = 0.6368\,\text{m}$
  - Solver output: $t_1 = 5221.7\,\text{s}$, $h_1 = 0.221\,\text{m}$

The results show a reasonable approximation of the tsunami's arrival time for both buoys. Still, the FWave solver underestimates the height of the simulated waves on both buoys, suggesting the FWave solver may dissipate wave energy more than expected as the tsunami propagates.

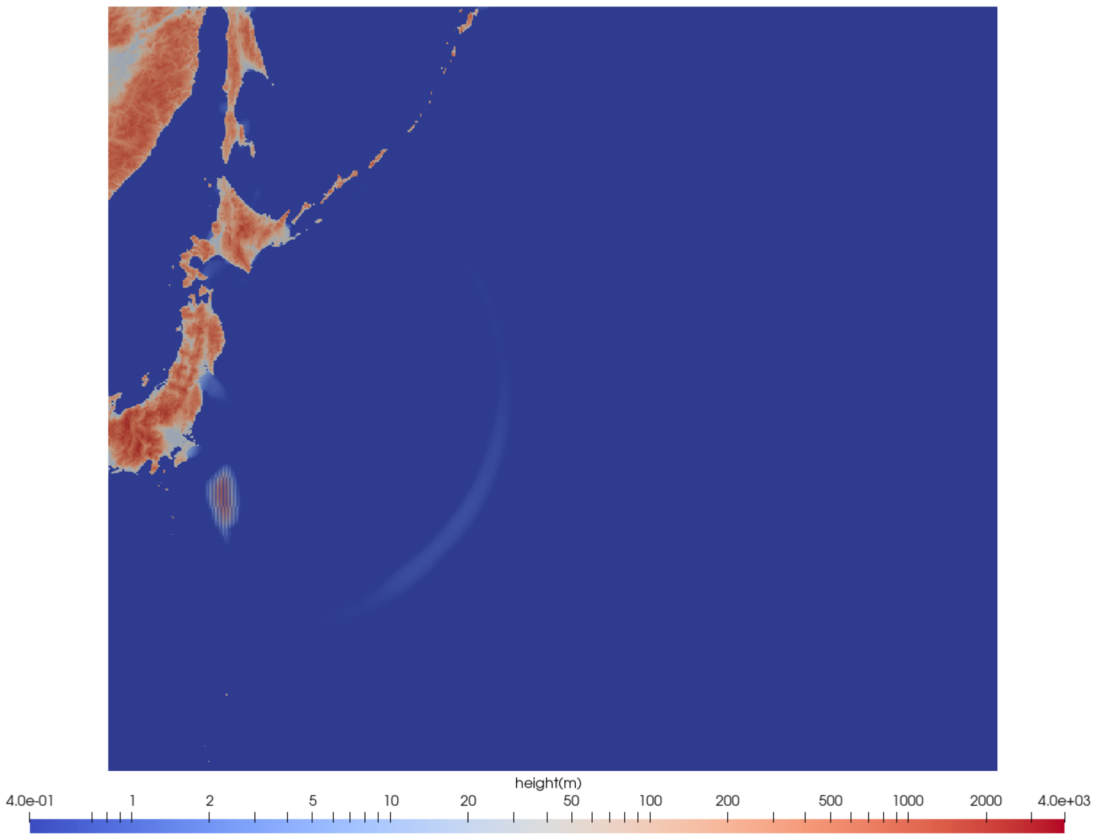


Figure 6.12.: Artifacts appearing in the simulation around $t = 3000\,\text{s}$ using the FWave solver.

Timestepping

To calculate the timesteps in finite volume solvers, we use the maximum wave speed of all the cells in the domain in a given timestep (this is described as the eigenvalue in the context of our solvers) and multiply this with a TSR factor to make sure no waves can ever overlap.

Due to how we implemented the applications in ExaHyPE 2, the domain is always divided into the same number of cells in x- and y-directions. Our domain has a length of 7000 kilometers in the x-direction and 4000 kilometers in the y-direction, resulting in rectangular-shaped cells instead of squares. Our timestep calculations described in Section 4.1.2 only consider the length of the cells in the x-direction when calculating the timesteps as it assumes the cells are square-shaped. Therefore, overlapping waves in the y-direction result in the artifacts observed in Figure 6.12.

These artifacts were prevented by adding a safety step after each iteration to fix the minimum cell size, overriding it with a minimum of the x- or y-direction. Alternatively, the domain is cut off from the non-critical areas on the east, resulting in a square-shaped domain and, therefore, solving the issue. Both of these measures were put in place for the following simulations.

HLLEM

Next, we simulate the initial Tōhoku tsunami using the HLLEM solver described in Chapter 4.1.2. The tsunami simulation exhibits a maximum wave speed of $\lambda_{max} \approx 310\,\mathrm{m/s}$ throughout the simulation, consistent with the FWave solver's results. However, the HLLEM solver shows a faster wave propagation, with the tsunami waves reaching the Soma shoreline at approximately $t \approx 1600\,\mathrm{s}$. This result is closer to the real-world observation of 9 minutes $t \approx 540\,\mathrm{s}$.

To further investigate the HLLEM solver's performance, we compared the simulated wave heights and arrival times at two available DART buoys, 21418 and 21419, from the National Data Buoy Center (NDBC) near the Japanese coast with real-world data and the results from the ExaHyPE 1 simulation, which are reported to be accurate in the literature [22]. This comparison was made using the probes implemented in Chapter 4.4:

- **Buoy 1:**
  - Real-world data: $t_0 \approx 30\,\mathrm{min} = 1800\,\mathrm{s}$, $h_0 \approx 2\,\mathrm{m}$
  - ExaHyPE 1 output: $t_0 = 1813.8\,\mathrm{s}$, $h_0 = 1.85232\,\mathrm{m}$
  - Solver output: $t_0 = 1775.683\,\mathrm{s}$, $h_0 = 1.52\,\mathrm{m}$

- **Buoy 2:**
  - Real-world data: $t_1 \approx 90\,\text{min} = 5400\,\text{s}$, $h_1 \approx 0.7\,\text{m}$
  - ExaHyPE 1 output: $t_1 = 5278.8\,\text{s}$, $h_1 = 0.6368\,\text{m}$
  - Solver output: $t_1 = 5135.071\,\text{s}$, $h_1 = 0.179\,\text{m}$

The HLLEM solver's results show a reasonable approximation of the tsunami's arrival times at both buoys, though the simulated wave heights are lower than those observed in reality. On both buoys, the HLLEM solver predicts the arrival time very close to the observed time. However, the wave height is significantly underpredicted, suggesting that the HLLEM solver may dissipate wave energy more than expected as the tsunami propagates.

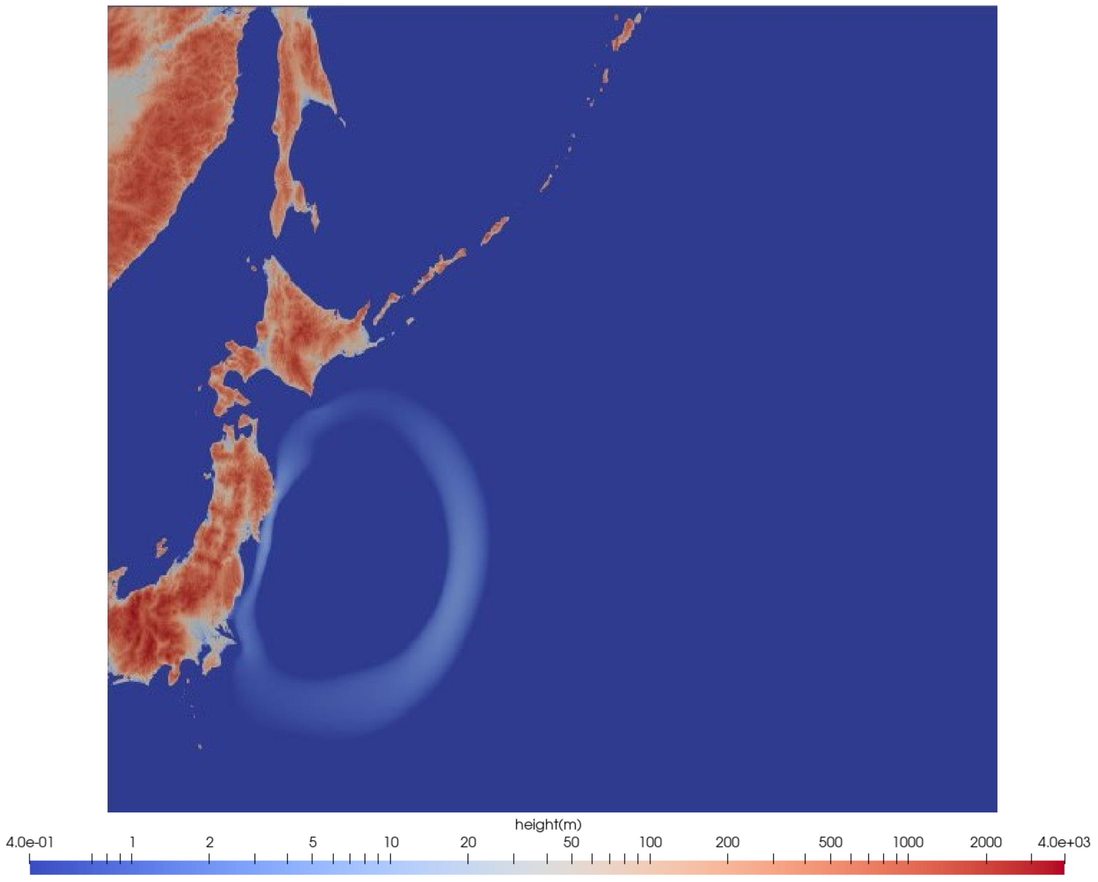


Figure 6.13.: Tōhoku tsunami approaching Soma at $t = 1607\,\text{s}$ displaying $h + b$ in logarithmic scale using the HLLEM solver.

Limiting Solver

Due to the issues with the ADER-DG solver in shallow ocean depths mentioned in Chapter 4.3, a limiting solver was implemented using the HLLEM solver on the shoreline where the water depth is less than 10 meters, while the ADER-DG solver is used in the deeper ocean. The resulting tsunami is shown in Figure 6.14. Throughout the simulation, the tsunami reaches a maximum wave speed of $\lambda_{max} \approx 290\,\mathrm{m/s}$. Although waves appear to dissipate near the shore due to the switch to HLLEM solver —and because our solver currently does not support the transfer of variables between solvers— the waves are expected to reach the shore around $t = 1000\,\mathrm{s}$, which still is shorter than the observed time of around $t = 540s$ after the earthquake. However, this result aligns with the ExaHyPE 1 simulation used in the study [21], where the output at $t = 900\,\mathrm{s}$ is shown in Figure 6.15.

In addition to these observations, we analyzed the wave heights and arrival times at two DART buoys (21418 and 21419) near the Japanese coast from the real-world data and the ExaHyPE 1 results [22] to further evaluate the performance of the ADER-DG solver. The comparison shows a higher degree of accuracy than the HLLEM solver:

- **Buoy 1:**
  - Real-world data: $t_0 \approx 30\,\mathrm{min} = 1800\,\mathrm{s}$, $h_0 \approx 2\,\mathrm{m}$
  - ExaHyPE 1 output: $t_0 = 1813.8\,\mathrm{s}$, $h_0 = 1.85232\,\mathrm{m}$
  - Solver output: $t_0 = 1814.062\,\mathrm{s}$, $h_0 = 2.05\,\mathrm{m}$
- **Buoy 2:**
  - Real-world data: $t_1 \approx 90\,\mathrm{min} = 5400\,\mathrm{s}$, $h_1 \approx 0.7\,\mathrm{m}$
  - ExaHyPE 1 output: $t_1 = 5278.8\,\mathrm{s}$, $h_1 = 0.6368\,\mathrm{m}$
  - Solver output: $t_1 = 5292.048\,\mathrm{s}$, $h_1 = 0.529\,\mathrm{m}$

The ADER-DG solver closely matches the ExaHyPE 1 simulation's arrival time at Buoy 1, with only about 0.26 seconds difference. The simulated wave height is slightly higher than the observed value, indicating a slight overestimation, but remains within an acceptable range.

Overall, the limited solver provides results that agree with the real-world data. The minor discrepancies in the wave height are expected to be caused by not using a smoothing function over the bathymetry data, as studies like [21] have reported reduced errors with such smoothing is applied.

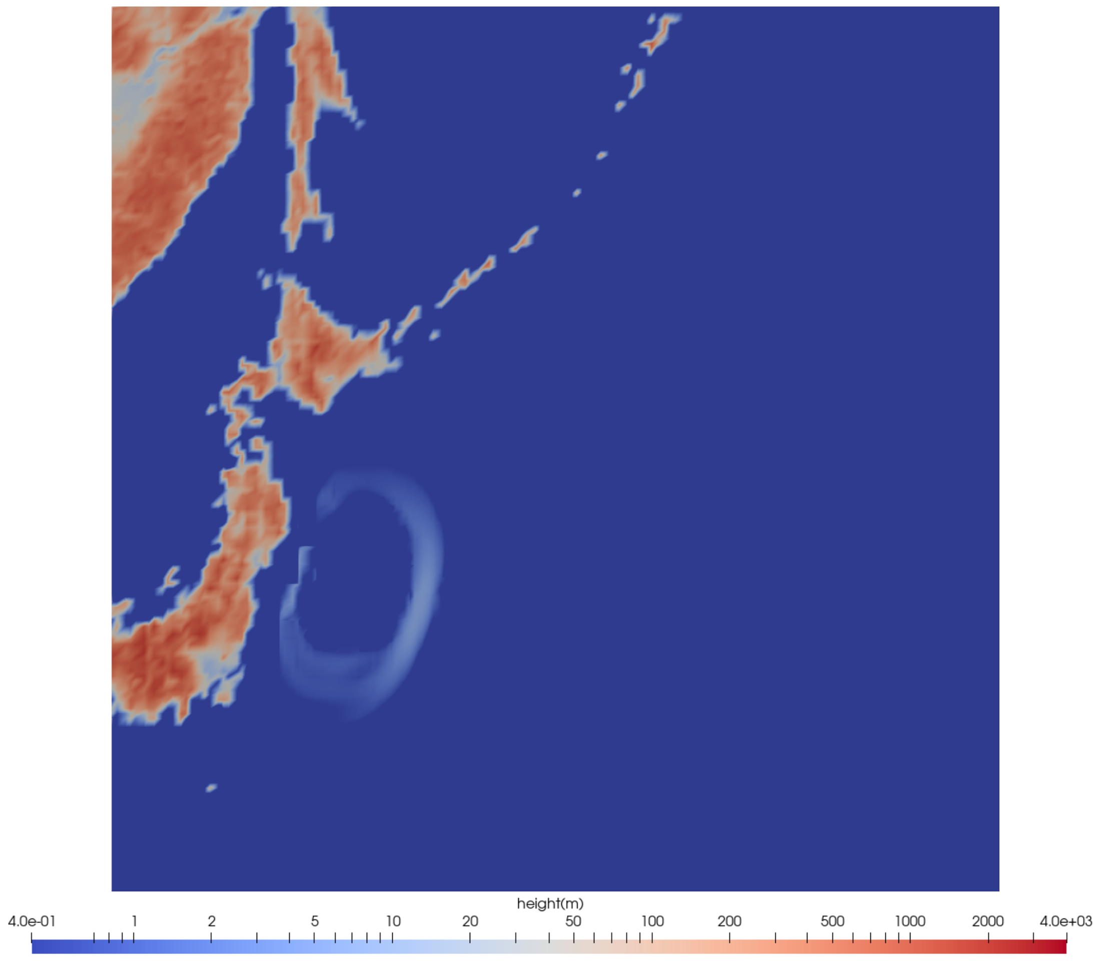


Figure 6.14.: Tōhoku tsunami approaching Soma at $t = 900\,\mathrm{s}$ displaying $h + b$ in logarithmic scale using the limiting solver.

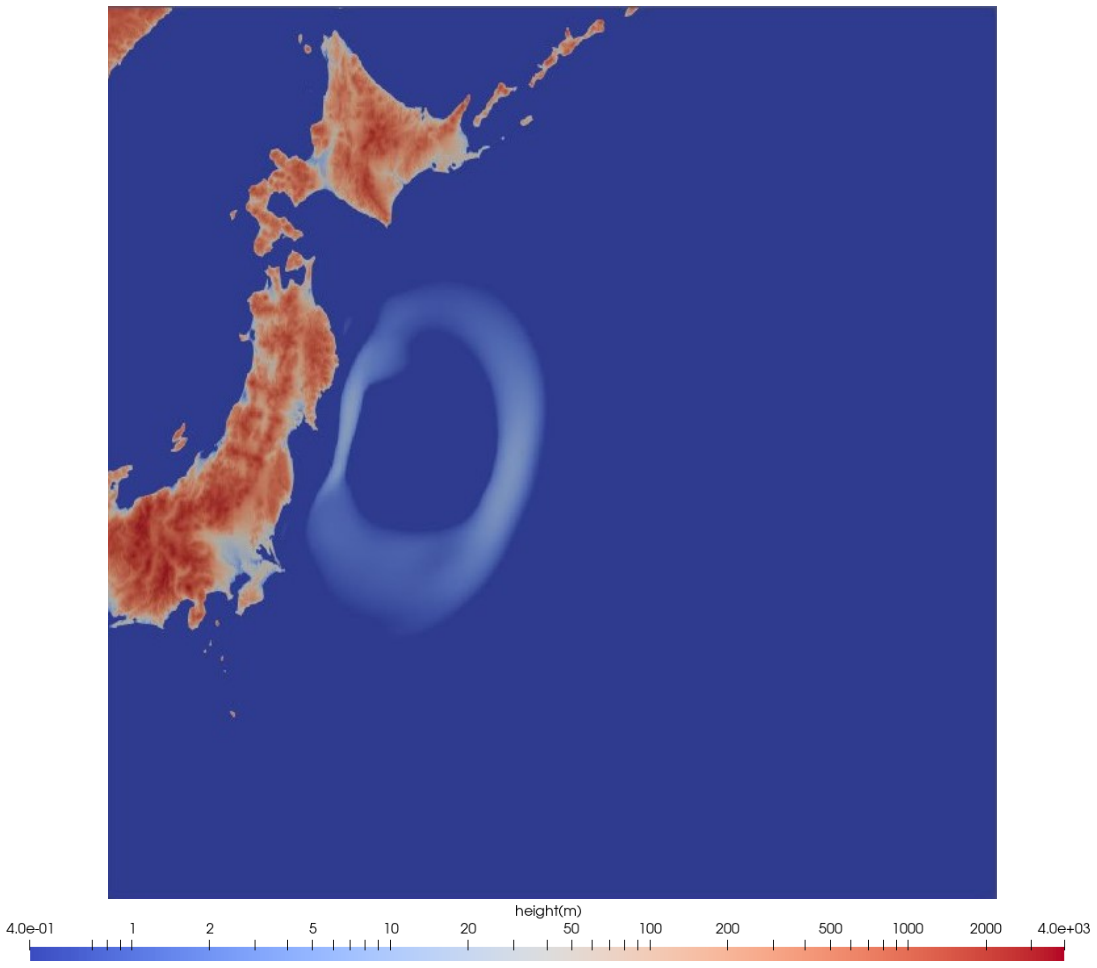


Figure 6.15.: Tōhoku tsunami approaching Soma at $t = 900\,\text{s}$ displaying $h + b$ in logarithmic scale using the limiting solver from ExaHyPE 1 with level 2 having Gaussian smoothing for the bathymetry data.

6.2. UM-Bridge with Metropolis-Hastings Markov Chain Monte Carlo

This section presents the results of the Metropolis-Hastings Markov Chain Monte Carlo (MHMCMC) algorithm, as implemented in the UM-Bridge interface described in Chapter 5. As our target function, we use wave heights and arrival times from two DART buoys (21418 and 21419) provided by the National Data Buoy Center (NDBC) near the Japanese coast during the 2011 Tōhoku tsunami.

The object is to estimate the tsunami's initial displacement parameter θ. In each iteration of the MHMCMC algorithm, a candidate θ^i is proposed and evaluated using our forward model to compute the likelihood function. The candidate is accepted or rejected based on the MHMCMC acceptance criteria, as detailed in Chapter 5.2.4.

Real-world measurements from the two buoys during the 2011 Tōhoku tsunami define the target distribution. We use a normal distribution $N(\mu, \Sigma)$ for our likelihood function, where the parameters are given as follows:

	μ	Σ
h_0	$1,85232$	$0,1$
h_1	$0,6368$	$0,1$
t_0	$1813,8$	$0,75$
t_1	$5278,8$	$0,75$

Table 6.1.: Mean (μ) and variance (Σ) for the normal distribution used in the likelihood function.

The algorithm, as described in [22] and outlined in Algorithm 8, was executed for 200 iterations, starting from an initial guess of $\theta^0 = (200000\,\text{m}, 200000\,\text{m})$.

Figure 6.16 shows the progression of the guessed θ^i points throughout the iterations of the UQ algorithm. Figure 6.17 depicts the convergence of the MHMCMC algorithm, illustrating how the average of the guessed points approach the actual initial displacement parameter θ for the tsunami.

The progression of the θ^i values throughout the MHMCMC iterations is promising. Figure 8 shows that the guessed θ^i points gradually converge to values closer to the initial parameter. The algorithm appears to converge towards a positive displacement value for θ, rather than the initially reported value from the paper [22].

This deviation is attributed to several factors impacting the accuracy of the output function y. Specifically, as detailed in Section 6.1.2, issues such as the decreased mesh depth leading to larger cells, resulting in bigger time steps, and the use of non-smoothed bathymetry data introduce noise into the simulations. These factors contribute to discrepancies between the estimated and the reported values of θ. Despite these challenges, the results indicate that the MHMCMC method effectively approximates the initial displacement, though with some variation.

It is also worth noting that the results could improve accuracy if the MHMCMC algorithm were replaced with the more advanced Markov Chain Monte Carlo method discussed in Chapter 5.2.4. The MLMCMC approach, with its ability to more efficiently explore the parameter space, would reduce the computational burden and potentially provide more accurate estimates with fewer samples, thereby addressing some of the limitations observed with the current MHMCMC method.

Overall, while the exact value of θ differs from the reported value, the results suggest that the model can identify a positive displacement parameter that is reasonably close to the expected initial condition, given the limitations of the current simulation setup.

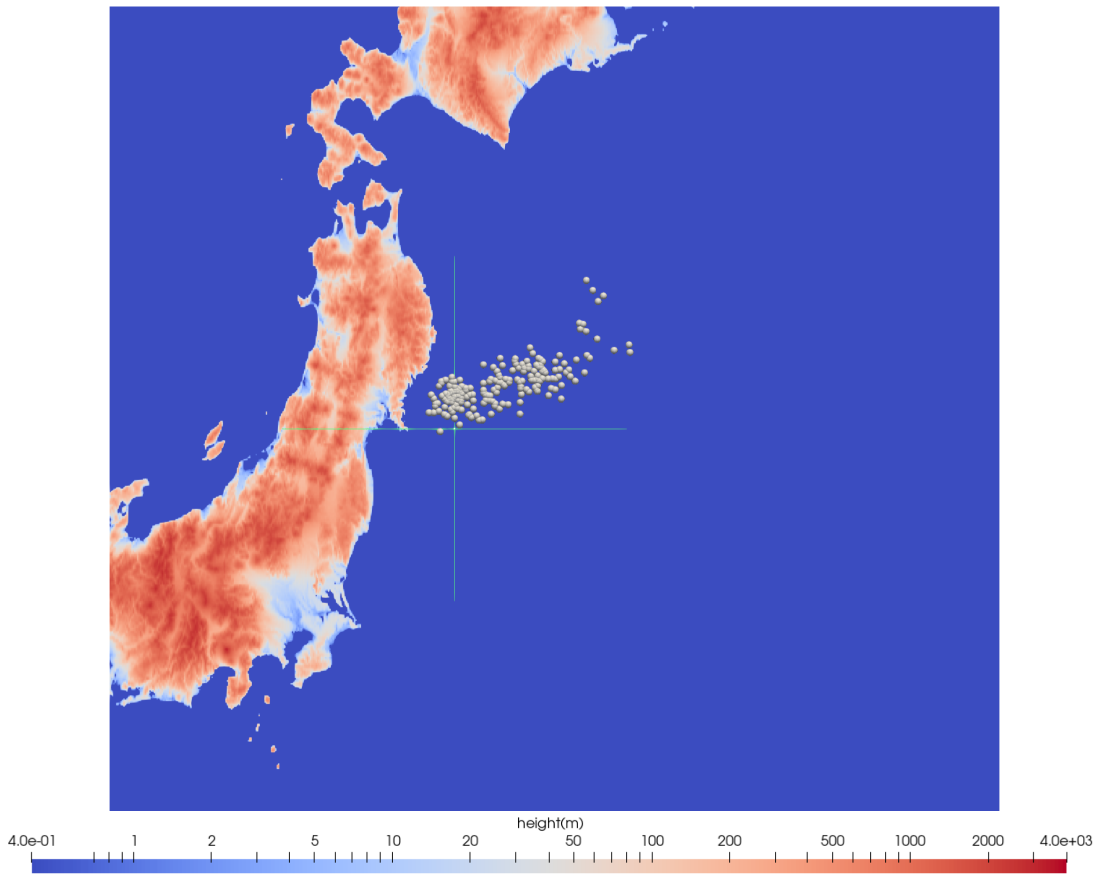


Figure 6.16.: The locations of θ^i for MHMCMC using the ADER-DG solver with min-depth 4 after 300 iterations.

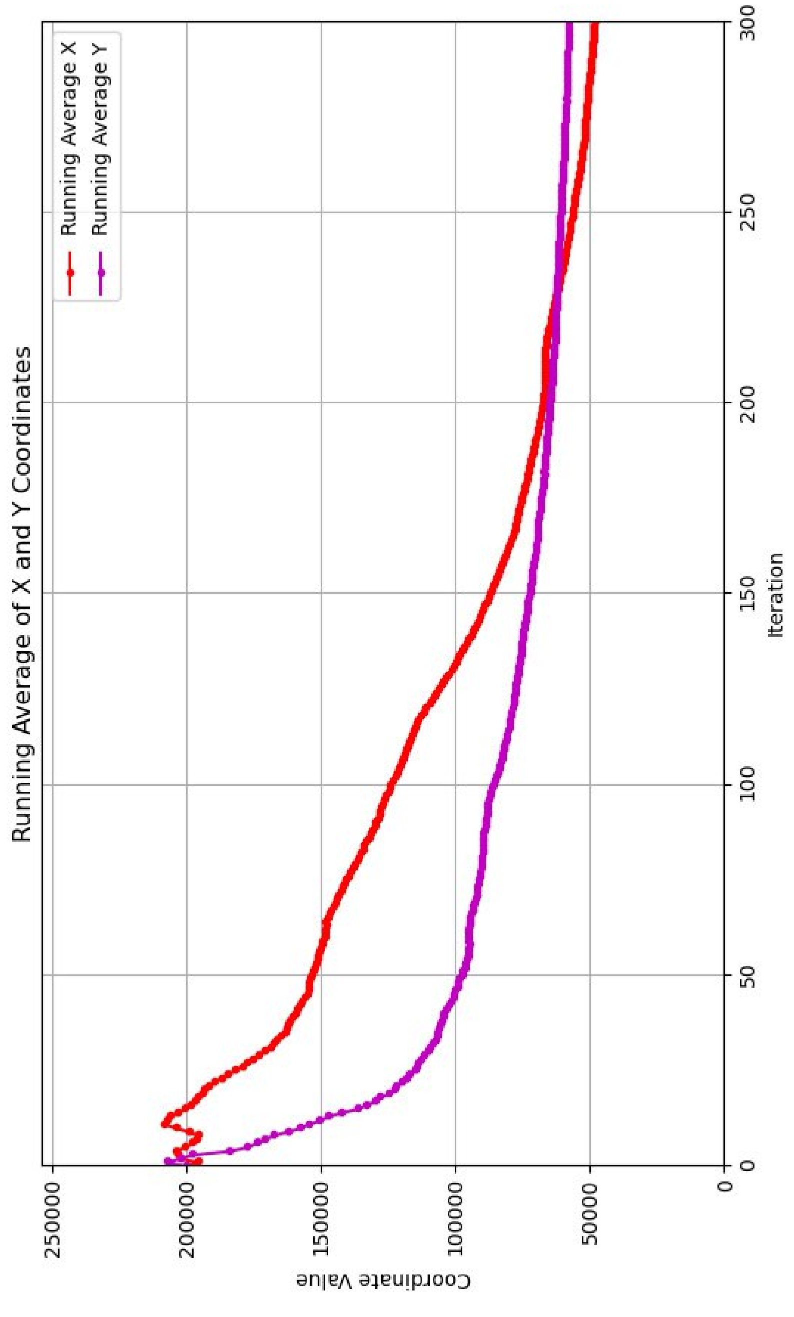


Figure 6.17.: MHMCMC convergence of the average values for X and Y where $\theta^i = (X, Y)$ over the 300 iterations.

7. Conclusion and Future Work

This thesis sets out to implement a reliable solver for shallow water equations using ExaHyPE 2 and a framework for uncertainty quantification algorithms (UM-Bridge) applied to tsunami modeling. The key findings show that the developed solver accurately modeled the Tōhoku tsunami, and the UM-Bridge framework was successfully coupled with the model, allowing the estimation of the initial conditions for the Tōhoku tsunami.

The results presented in this thesis contribute significantly to the field of tsunami modeling by allowing utilization of ExaHyPE 2's scaling capabilities and ability to handle large-scale simulations in future models. Integrating UM-Bridge enables easier development of future uncertainty quantification algorithms using other models within ExaHyPE 2.

However, the study involved several simplifications in tsunami modeling and the uncertainty quantification algorithms. These simplifications, such as non-smoothed bathymetry data, may have caused inaccuracies in the model. Additionally, the computationally intensive Metropolis-Hastings Markov Chain Monte Carlo algorithm was chosen for its simple implementation, but its resource requirements should be considered in practical applications.

In addition to improving upon the challenges mentioned above, future research could further develop the UM-Bridge framework extending its applicability to other scenarios and models as well as implement alternative algorithms, such as the Multilevel Markov Chain Monte Carlo algorithm discussed in Chapter 5.2.4, to enhance the results.

In conclusion, this thesis marks a significant step forward in tsunami simulations and uncertainty quantification within ExaHyPE 2. The successful implementation of a reliable solver for shallow water equations and the integration of the UM-Bridge framework for uncertainty quantification in tsunami modeling has the potential to significantly advance the field, providing more accurate and reliable models for tsunami events.

List of Figures

4.1. The split of the cells for the limiting solver with min depth 4. For the yellow domain HLLEM FV solver is used, while for the blue domain the ADER-DG solver is used. 16

5.1. The UM-Bridge interface 17
5.2. The UM-Bridge sequence diagram 19

6.1. Initial conditions for the dam break scenario 24
6.2. 2D-surface plot over time for dam break scenario with the generic Riemann solver. 25
6.3. 2D-surface plot over time for dam break scenario with the FWave solver. 26
6.4. 2D-surface plot over time for dam break scenario with the HLLEM solver. 27
6.5. 2D-surface plot over time for dam break scenario with the ADER-DG solver. 28
6.6. Line plot of the height along the x-axis comparing results from different solvers in timesteps $t = 0\,\mathrm{s}$ and $t = 0.1\,\mathrm{s}$. 29
6.7. Line plot of the height along the x-axis comparing results from different solvers in timesteps $t = 0.5\,\mathrm{s}$ and $t = 1.0\,\mathrm{s}$. 30
6.8. Line plot of the absolute value of momentum in the x-direction along the x-axis comparing results from different solvers in timesteps $t = 0\,\mathrm{s}$ and $t = 0.1\,\mathrm{s}$. 31
6.9. Line plot of the absolute value of momentum in the x-direction along the x-axis comparing results from different solvers in timesteps $t = 0.5\,\mathrm{s}$ and $t = 1.0\,\mathrm{s}$. 32
6.10. Initial conditions of Tōhoku tsunami displaying $h + b$ in logarithmic scale. 34
6.11. Tōhoku tsunami approaching Soma at $t = 1607\,\mathrm{s}$ displaying $h + b$ in logarithmic scale using the FWave solver with TSR factor of 0.4. 35
6.12. Artifacts appearing in the simulation around $t = 3000\,\mathrm{s}$ using the FWave solver. 36
6.13. Tōhoku tsunami approaching Soma at $t = 1607\,\mathrm{s}$ displaying $h + b$ in logarithmic scale using the HLLEM solver. 38
6.14. Tōhoku tsunami approaching Soma at $t = 900\,\mathrm{s}$ displaying $h + b$ in logarithmic scale using the limiting solver. 40

6.15. Tōhoku tsunami approaching Soma at $t = 900\,\mathrm{s}$ displaying $h + b$ in logarithmic scale using the limiting solver from ExaHyPE 1 with level 2 having Gaussian smoothing for the bathymetry data. 41
6.16. The locations of θ^i for MHMCMC using the ADER-DG solver with min-depth 4 after 300 iterations. 43
6.17. MHMCMC convergence of the average values for X and Y where $\theta^i = (X, Y)$ over the 300 iterations. 44

List of Tables

6.1. Mean (μ) and variance (Σ) for the normal distribution used in the likelihood function. 42

Code listings

A.1. Initial Dambreak Scenario. 53
A.2. Flux for FV and ADER-DG Solvers. 54
A.3. Eigenvalues for FV Solvers. 55
A.4. Max eigenvalue for ADER-DG Solver. 55
A.5. NCPs for ADER-DG Solver 56

B.1. Server for UM-Bridge 57
B.2. Client for UM-Bridge 59

Bibliography

[1] J. Adler and S. Kurbiel. "Markov Chain Monte Carlo." In: *WiSt - Wirtschaftswissenschaftliches Studium* 44.5 (2015), pp. 238–245. ISSN: 0340-1650.

[2] D. S. 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." In: *SIAM Journal on Scientific Computing* 24.3 (2003), pp. 955–978. DOI: `10.1137/S106482750139738X`. eprint: `https://doi.org/10.1137/S106482750139738X`.

[3] F. Benkhaldoun and M. Seaïd. "A simple finite volume method for the shallow water equations." In: *Journal of Computational and Applied Mathematics* 234.1 (2010), pp. 58–72. ISSN: 0377-0427. DOI: `https://doi.org/10.1016/j.cam.2009.12.005`.

[4] M. Dashti and A. M. Stuart. "The Bayesian Approach To Inverse Problems." In: (2013).

[5] M. Dumbser, D. S. Balsara, E. F. Toro, and C.-D. Munz. "A unified framework for the construction of one-step finite volume and discontinuous Galerkin schemes on unstructured meshes." In: *Journal of Computational Physics* 227.18 (2008), pp. 8209–8253. ISSN: 0021-9991. DOI: `https://doi.org/10.1016/j.jcp.2008.05.025`.

[6] B. Einfeldt, C. Munz, P. Roe, and B. Sjögreen. "On Godunov-type methods near low densities." In: *Journal of Computational Physics* 92.2 (1991), pp. 273–295. ISSN: 0021-9991. DOI: `https://doi.org/10.1016/0021-9991(91)90211-3`.

[7] K. S. Erduran, V. Kutija, and C. J. M. Hewett. "Performance of finite volume solutions to the shallow water equations with shock-capturing schemes." In: *International Journal for Numerical Methods in Fluids* 40.10 (2002), pp. 1237–1273. DOI: `https://doi.org/10.1002/fld.402`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/fld.402`.

[8] L. C. Evans. *Partial Differential Equations*. American Mathematical Society, 1983.

[9] U. Gomez, G. B. Gadeschi, and T. Weinzierl. *GPU Offloading in ExaHyPE Through C++ Standard Algorithms*. 2023. arXiv: `2302.09005 [cs.MS]`.

[10] J. S. Hesthaven and T. Warburton. *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*. 1st. Springer Publishing Company, Incorporated, 2007. ISBN: 0387720650.

[11] K.-R. Koch. "Monte Carlo methods." In: *GEM - International Journal on Geomathematics* 9.1 (2018), pp. 117–143.

[12] C. Kühbacher. *Shallow Water Derivation and Applications*. 2009. URL: `https://wwwold.mathematik.tu-dortmund.de/lsiii/cms/papers/Kuehbacher2009.pdf` (visited on 08/03/2024).

[13] R. J. LeVeque, D. L. George, and M. J. Berger. "Tsunami modelling with adaptively refined finite volume methods." In: *Acta Numerica* 20 (2011), pp. 211–289. DOI: `10.1017/S0962492911000043`.

[14] C. M. Loi, H. Bockhorst, and T. Weinzierl. *SYCL compute kernels for ExaHyPE*. 2023. arXiv: `2306.16731 [cs.MS]`.

[15] National Data Buoy Center. *NDBC: National Data Buoy Center*. `https://www.ndbc.noaa.gov/`. Accessed: 2024-08-10. 2024.

[16] Z. Niu, A.-A. Gabriel, L. Seelinger, and H. Igel. *Modeling and Quantifying Parameter Uncertainty of Co-seismic Non-classical Nonlinearity in Rocks*. 2023. arXiv: `2306.04197 [physics.geo-ph]`.

[17] L. Rannabauer, M. Dumbser, and M. Bader. "ADER-DG with a-posteriori finite-volume limiting to simulate tsunamis in a parallel adaptive mesh refinement framework." In: *Computers Fluids* 173 (2018), pp. 299–306. ISSN: 0045-7930. DOI: `https://doi.org/10.1016/j.compfluid.2018.01.031`.

[18] A. Reinarz, D. E. Charrier, M. Bader, L. Bovard, M. Dumbser, K. Duru, F. Fambri, A.-A. Gabriel, J.-M. Gallard, S. Köppel, L. Krenz, L. Rannabauer, L. Rezzolla, P. Samfass, M. Tavelli, and T. Weinzierl. "ExaHyPE: An engine for parallel dynamically adaptive simulations of wave problems." In: *Computer Physics Communications* 254 (2020), p. 107251. ISSN: 0010-4655. DOI: `https://doi.org/10.1016/j.cpc.2020.107251`.

[19] H. Schulz, G. B. Gadeschi, O. Rudyy, and T. Weinzierl. "Task inefficiency patterns for a wave equation solver." In: *CoRR* abs/2105.12739 (2021). arXiv: `2105.12739`.

[20] L. Seelinger, V. Cheng-Seelinger, A. Davis, M. Parno, and A. Reinarz. "UM-Bridge: Uncertainty quantification and modeling bridge." In: *Journal of Open Source Software* 8.83 (2023), p. 4748. DOI: `10.21105/joss.04748`.

[21] L. Seelinger, A. Reinarz, M. B. Lykkegaard, R. Akers, A. M. A. Alghamdi, D. Aristoff, W. Bangerth, J. Bénézech, M. Diez, K. Frey, J. D. Jakeman, J. S. Jørgensen, K.-T. Kim, M. Martinelli, M. Parno, R. Pellegrini, N. Petra, N. A. B. Riis, K. Rosenfeld, A. Serani, L. Tamellini, U. Villa, T. J. Dodwell, and R. Scheichl. *Democratizing Uncertainty Quantification*. 2024. arXiv: `2402.13768 [cs.MS]`.

[22] L. Seelinger, A. Reinarz, L. Rannabauer, M. Bader, P. Bastian, and R. Scheichl. *High Performance Uncertainty Quantification with Parallelized Multilevel Markov Chain Monte Carlo*. 2021. arXiv: `2107.14552 [cs.MS]`.

[23] E. Toro. "Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction." In: Jan. 2009. DOI: `10.1007/b79761`.

[24] M.-C. Trinh and T. Mukhopadhyay. "Semi-analytical atomic-level uncertainty quantification for the elastic properties of 2D materials." In: *Materials Today Nano* 15 (2021), p. 100126. ISSN: 2588-8420. DOI: `https://doi.org/10.1016/j.mtnano.2021.100126`.

[25] T. Weinzierl. "The Peano software—parallel, automaton-based, dynamically adaptive grid traversals." In: *ACM Transactions on Mathematical Software* 45.2 (2019), p. 14.

[26] M. Wille, T. Weinzierl, G. Brito Gadeschi, and M. Bader. "Efficient GPU Offloading with OpenMP for a Hyperbolic Finite Volume Solver on Dynamically Adaptive Meshes." In: *High Performance Computing*. Ed. by A. Bhatele, J. Hammond, M. Baboulin, and C. Kruse. Cham: Springer Nature Switzerland, 2023, pp. 65–85. ISBN: 978-3-031-32041-5.

[27] L. Yan and L. Guo. "Stochastic Collocation Algorithms Using l_1-Minimization for Bayesian Solution of Inverse Problems." In: *SIAM Journal on Scientific Computing* 37.3 (2015), A1410–A1435. DOI: `10.1137/140965144`. eprint: `https://doi.org/10.1137/140965144`.

A. Implementation of SWE in ExaHyPE 2

This appendix details the implementation of the shallow water equations (SWE) application in ExaHyPE 2. The SWE application has one main script - `swe.py`- and various scenario scripts such as `radial-dam-break/radial_dam_break.py` in their respective directories. These scenario scripts modify the configuration settings of `swe.py` to tailor the simulations to specific conditions.

Scenario Script:

Each scenario script sets up the initial and boundary conditions defined in their respective `SWE.cpp` file. Below is an example of how the initial conditions for a radial dam break scenario are defined:

Listing A.1: Initial Dambreak Scenario.

```

void applications::exahype2::swe::SWE::initialCondition(
  double* __restrict__ Q,
  const ::tarch::la::Vector<Dimensions, double>& x,
  const ::tarch::la::Vector<Dimensions, double>& h,
  bool gridIsConstructed
) {
  for (int i = 0; i < NumberOfUnknowns + NumberOfAuxiliaryVariables; i++) {
    Q[i] = 0.0;
  }

  const double domainSizeHalfX = DomainSize[0] / 2.0;
  const double domainSizeHalfY = DomainSize[1] / 2.0;
  const double distanceFromOrigin = sqrt(
    (x[0] - domainSizeHalfX) * (x[0] - domainSizeHalfX)
    + (x[1] - domainSizeHalfY) * (x[1] - domainSizeHalfY)
  );
  Q[s::h] = distanceFromOrigin <= DAM_RADIUS
              ? INITIAL_WATER_HEIGHT_INSIDE_DAM
              : INITIAL_WATER_HEIGHT_OUTSIDE_DAM;
}
```

This function initializes the water height, momentums and bathymetry of the cells in the domain.

Flux Implementation for the Solvers:

The following code snippets demonstrate how the fluxes are computed for both FV and ADER-DG solvers.

Listing A.2: Flux for FV and ADER-DG Solvers.

```

void applications::exahype2::swe::flux(
  const double* __restrict__ Q,
  const tarch::la::Vector<Dimensions, double>& x,
  const tarch::la::Vector<Dimensions, double>& h,
  double t,
  double dt,
  int normal,
  double* __restrict__ F
) {
  using s = VariableShortcuts
  double ih = 1.0 / std::max(Q[0], 1e-2);

    F[0] = Q[1 + normal];
    F[1] = Q[1 + normal] * Q[1] * ih;
    F[2] = Q[1 + normal] * Q[2] * ih;
    F[3] = 0.0;

  }
}
```

Eigenvalue Implementation for the Solvers:
The following code snippets show how the eigenvalues are calculated for both FV and ADER-DG solvers:

Listing A.3: Eigenvalues for FV Solvers.

```

void applications::exahype2::swe::eigenvalues(
  const double* __restrict__ Q,
  const tarch::la::Vector<Dimensions, double>& x,
  const tarch::la::Vector<Dimensions, double>& h,
  double t,
  double dt,
  int normal,
  double* __restrict__ L
) {
  using s = VariableShortcuts
  L[0] = u_n + c;
  L[1] = u_n - c;
  L[2] = u_n;

}
```

Listing A.4: Max eigenvalue for ADER-DG Solver.

```

double applications::exahype2::swe::maxEigenvalue(
  const double* __restrict__ Q,
  const tarch::la::Vector<Dimensions, double>& x,
  const tarch::la::Vector<Dimensions, double>& h,
  double t,
  double dt,
  int normal
) {
    const double u = Q[1 + normal] / std::max(Q[0], 1e-2);
    const double c = std::sqrt(grav * std::max(Q[0], 1e-2));

    return std::max(std::abs(u + c), std::abs(u - c));
}
```

Non-Conservative Product Implementation for the ADER-DG Solver:
This snippet shows how the non-conservative products for the ADER-DG solver are calculated:

Listing A.5: NCPs for ADER-DG Solver

```


void ::applications::exahype2::swe::AbstractaderSolver::nonconservativeProduct(
  const double*__restrict__ Q,
  const double*__restrict__ deltaQ,
  const tarch::la::Vector<Dimensions, double> &faceCentre,
  const tarch::la::Vector<Dimensions, double> &volumeH,
  double t,
  double dt,
  int normal,
  double*__restrict__ BTimesDeltaQ)
{
    for (int i = 0; i < NumberOfUnknowns + NumberOfAuxiliaryVariables; i++) {
        BTimesDeltaQ[i] = 0.0;
    }
    BTimesDeltaQ[normal + 1] = grav * std::max(Q[0], 1e-2) * (deltaQ[0] + deltaQ
        [3]);

}
```

B. Implementation of UM-Bridge in ExaHyPE 2

The UM-Bridge framework is used to couple the SWE solver with uncertainty quantification algorithms, enabling the estimation of initial conditions based on observed data. Below are snippets of the server and client implementations.

Server Implementation:

The server script defines a `TsunamiModel` that runs the SWE solver and returns the output $y = G(\theta)$ explained in more detail in Chapter 5.

Listing B.1: Server for UM-Bridge

```python
class TsunamiModel(umbridge.Model):
    def __call__(self, parameters, config):
        application_name = "ExaHyPE2-SWE-LimitingGlobalAdaptive-Release"
        input_data = "%f\n%f\n%f\n%f" % (parameters[0][0], parameters[0][1],
            parameters[0][0], parameters[0][1])
        run(["./" + application_name], input=input_data.encode(), # Provide
            input data as bytes)
        operating_directory = "./solution"
        # read values from csv file
        csv_frame = pd.concat([pd.read_csv(csv_file, skipinitialspace=True,
            header=0, names=["n1", "n2", "t", "x1", "x2", "h", "hu", "hv", "b"])
             for csv_file in glob("tracers/"+"*.csv")], ignore_index=True)
        print("Reading finished, now formatting dataframe.")
        csv_frame['coordinates'] = csv_frame.apply(lambda row: (row['x1'], row['
            x2']), axis=1)
        csv_frame = csv_frame.drop(["x1", "x2", "n1", "n2"], axis=1)
        csv_frame = csv_frame.iloc[csv_frame.groupby(['coordinates'])['h'].
            idxmax()]
        # return output vector
        output = []
        for _, row in csv_frame.iterrows():
            output.append(row['t'])
            output.append(row['h'] + row['b'])
        print("Output: " + str(output))
        return [output]
```

```
model = TsunamiModel()
umbridge.serve_models([model], 4242)
```

Client Implementation:
The client script connects to the server using HTTP to send input parameters and get model output.

Listing B.2: Client for UM-Bridge

```
model = umbridge.HTTPModel("http://0.0.0.0:4242", "forward")
model.get_input_sizes(config)
model.get_output_sizes(config)
desired_output = [[1813.8, 1.85232, 5278.8, 0.6368]]
print(desired_output)
input_x = 200000
input_y = 200000
output = model([[input_x, input_y]], config)
#Implementation of the chosen algorithm
```