# TIME-ADAPTIVE GRID HYDROLOGICAL SOLVER USING PETSC

## THESIS REPORT

MUKKUND SUNJII BABU GHEETHAA

SIM·CI

TU Delft

# Time-Adaptive Grid Hydrological Solver using PETSc

## Thesis Report

by

Mukkund Sunjii Babu Gheethaa

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday July 16, 2020 at 10:00 AM.

| | | |
|---|---|---|
| Student number: | 4817303 | |
| Project duration: | October 1, 2019 – July 1, 2020 | |
| Thesis committee: | Dr. Domenico Lahaye, | TU Delft, supervisor |
| | Prof. dr. Chris J. Budd, | University of Bath |
| | Dr. Matthias Möller, | TU Delft |
| | Ir. Eelco Naarding, | SIM-CI BV |

*This thesis is confidential and cannot be made public until July 16, 2020.*



TU Delft

# Preface

The thesis is an extension to my internship project conducted at SIM-CI, The Hague on July, 2019 as part of my master's program. A serial version of the model was successfully implemented and subsequently integrated into the company's software pipeline. Nevertheless, the computational overhead of the solver proved to be a bottleneck. Therefore, the need to parallelise the model emerged as a necessity. Furthermore, as an attempt to contribute to the open-source scientific software community, this thesis also is geared towards serving that purpose on top of providing a reliable hydrological model for the use of urban planning organisations.

In truth, this project wouldn't have been possible without the strong commitment and dedication offered from several of my support groups. Firstly, I would like to thank my mother who has provided with all the love and support that a son can ask for. She has and always will be the most important role-model in my life.

I would also like to sincerely pass my gratitude to my thesis supervisor, Prof. Domenico Lahaye for his wonderful advice and support throughout the course of this project. Finally, I would be remiss if I did not acknowledge the invaluable guidance and knowledge offered by Eelco Naarding and Martijn Stroeven of SIM-CI.

Thank you for your unwavering support.

*Mukkund Sunjii Babu Gheethaa*
*Delft, July 2020*

# Contents

# 1 Introduction

## 1.1. Motivation

At the start of the new millennium, countless studies have identified climate change as one of the eminent dangers posed to modern civilization. With the increase in the development of public infrastructure and the ever-expanding urbanization efforts, the rate of climate change is superseding any naturally evolving process. One of the many potentially disastrous consequences is the rising sea levels and the uncontrolled flooding of urban infrastructure.

Modern municipalities and governmental bodies are rushing to gather efficient methodologies to be better prepared for this inevitable consequence of flooding. Furthermore, financial institutions such as insurance firms must account for floods in their risk assessments. The accuracy of these estimates could potentially reflect the value of their respective investments into various critical infrastructure. A sound judgement on the reliability of the systems under extreme conditions such as heavy rainfalls and/or flash floods is an important factor to be considered.

Figure 1.1: Floods in York, UK

Accurate and efficient methodologies must be made accessible to these organisations to build smarter cities that adapt to a rapid climatic shift. Analysis of the overland flooding patterns can be provided by empirical studies on scaled-down models. The pitfalls of this method lie in the physical time and cost associated with building and maintaining experimental models. On the other end of the spectrum, it can be provided by numerical models governed by the physical laws of hydrology. With the advent of efficient high-performance computers, numerous governments rely on simulation data provided by numerical models.

Conventional hydrological numerical models are based on fixed grid methods. The conditions related to a typical flooding scenario are highly irregular because of the characteristics of the urban landscape and of the time-dependent flow patterns acquired by the flood. To provide a reliable and accurate solution, the numerical model must be efficient and accurately provide propagation information of the water. This is critical in predicting the overall damage inflicted by an overland flooding event. Fixed grid solvers are not capable of efficiently carrying out this task without the use of an unrealistic amount of computational resources.

Time-adaptive grid methods for hydrological applications provide a viable solution. Due to the adaptability of the method, a vast array of scenarios can be simulated with a high degree of accuracy while at the same time adhering to manageable grid sizes. A massively parallel solver further expedites the process by taking advantage of the high-performance computing resources available. Additionally, the use of an open source framework to base such a solver with the required documentation is needed by both academia and industry alike. After conducting a comprehensive survey of the available open-source, top-level scientific libraries, it was decided to base the model using PETSc (Portable Extensible Toolkit for Scientific Computation) [8]. The objective is thus to create a parallel, time-adaptive grid hydrological solver based on an open-source framework provided by the scientific library PETSc.

Due to the extensive nature of PETSc, the process of creating the final model several preliminary and intermediate steps. These steps included identifying the functions and capabilities of the various subsystems making up the scientific library. With the information in hand, the final model was created and validated with previously verified empirical and analytical cases. Furthermore, the various tuning parameters associated with a time-adaptive grid model is analysed to identify their effect on the accuracy and the computational resources required.

## 1.2. Involvement of SIM-CI Holding BV

SIM-CI, The Hague, The Netherlands is a tech startup that offered a simulation platform through a SaaS platform. Their flagship service was the simulation of cascading effects on the critical infrastructures of a city. Their software thus conducted a simulation of rare-events such as telecommunication failures, electrical outages, flooding events, or disruption of public transports in cities with typical modern amenities. Through the simultaneous operation of multiple models, the cascading effects of a given disturbance can be visualized through a Unity client offered by the company. The same can be achieved in the case of a dyke break or flash floods. Using specialised models that calculate the damage on buildings and the implications of a power outage or a telecommunication failure, the total financial impact of such an event can be reported to the user.

An in-house hydrological model proved to be a vital addition to the product's repertoire. Furthermore, the hydrological models could also be used to enrich the predictions calculated by the damage models. Therefore, the inception of the project was carried out in collaboration with SIM-CI [25]. Part of the primary objective of the thesis revolved around the functional requirements put forth by the development team of the company. According to the requirements, the final model must thus fulfill the following:

- Model Performance: The hydrological model must be robust and capable of handling a wide variety of scenarios including being able to model dyke breaks, and heavy rainfalls. Furthermore, the model must be scalable and written in C/C++.
- Accuracy of the Results : As the output of the model is used to estimate the damage, control the traffic flow, and/or the operation of first responders, the accuracy of the model is of high priority.
- Maintainability: The piece of software must be well documented and highly modular in nature. This could allow for easier debugging and for retrofitting the model to accommodate client-specific cases.
- Integration with the Software : In order to model cascading effects, the hydrological model must be integrated into the software pipeline. Therefore, the model must be built to suit the API adopted and maintained by the company. This indeed dictates the choice of the coding patterns of the model.
- Licensing: The final model created would henceforth be the intellectual property of the company. Therefore, the model must be based on open-source software that offers the appropriate distribution rights.

During the initial stages of the project, due to the passing of certain legal legislatures, SIM-CI was forced to stop all the ongoing operations by its parent company. *As a result, it was decided to carry out the project without the involvement of SIM-CI*. Nevertheless, valuable technical assistance was offered by Martijn Stroeven, CTO and Eelco Naarding, Scientific Software Programmer of SIM-CI throughout the duration of the project. With the fall-out of the collaboration with the company, it was found that the core of the functional requirements still provided value to the open-source scientific community. Therefore, together with the administration of SIM-CI and Dr. Lahaye (thesis supervisor), it was decided that the model created in this project not made proprietary. The source code for the models can be found in the author's Github repository [26].

## 1.3. Modelling Approach

In recent years, significant research has been done into numerically modelling overland flows. A majority of occurrences of free surface water flow in a natural setting can be captured by solving the two-dimensional shallow water equations (SWE). The SWE are obtained using the depth-averaged version of the Navier-Stokes equations. The equations assume a hydro-static condition across the vertical gradients of pressure. Furthermore, it maintains that the horizontal velocity is constant across the entire depth of the fluid section. Some of the numerical methods that were considered to solve the hyperbolic partial differential equation (PDE) are described in the following sub-sections.

### 1.3.1. Finite Element Method

The finite element method is a popular approach to solve PDE's. It is capable of tackling complex boundary conditions as opposed to a finite difference or a finite volume method. Behrens [10] followed a similar approach by employing a lower order finite element approach while carrying out local refinements to the mesh. For the temporal discretisation, a semi-implicit Lagrangian method was used. The interpolation and velocity calculations involved in the method are independent and hence are in an excellent position for parallelization. Similar flexibility is offered when the Discontinuous Galerkin (DG) method is combined with an adaptivity strategy that modifies the local order of accuracy in the areas of interest [55]. However, it was found that the number of degrees of freedom associated with DG methods is higher when compared to continuous finite element methods [21]. Additionally, the finite element approach poses a higher degree of difficulty in terms of implementation and optimization for scalability.

### 1.3.2. Finite Volume Method

On the other hand, a number of commercial codes are based on the finite volume method (FVM). These algorithms include the high-resolution Godunov-type methods which are very well suited to solve for the SWE. For instance, Leveque [41] uses a second-order accurate method to solve for various flooding test cases. Discontinuities in the domain, often called hydraulic jumps, are regular artifacts of overland flooding and can be solved effectively using high-resolution methods. The Godunov method can be applied by solving the Riemann problem across the domain. However, using an exact Riemann solver is computationally expensive. Therefore, an approximate Riemann solver offers an alternative solution to reduce the computational overhead. Liang [44] uses an HLLC approximate Riemann solver in conjunction with a Godunov type method. Additionally, to avoid spurious, non-physical oscillations usually observed in second-order methods, limiters can be applied to damp them down. Such an approach is adopted by Liang [45] to obtain a robust model capable of solving for dam-break scenarios. In the face of a number of benefits offered by FVM, it was decided to base the models presented in the project using this numerical methodology.

### 1.3.3. Grid Adaptivity

In the applications of geophysical modelling, the domain to be solved usually consists of vasts areas of land and water bodies. If one were to use a uniform grid solver for such applications, they would be heavily restricted to using low-resolution grids. Complex behaviours of over-surface flows such as the interaction with urban terrains such as buildings and canals cannot be resolved and hence, naturally lead to high localized errors. Couple of methodologies are available to overcome the grid resolution issue and increase the overall efficiency of the model.

The first solution is to use non-uniform grids which can be selectively refined in certain areas of

the grid. For instance, in tsunami modelling such a necessity is evident along the coastlines where the resolution of the grid is required to be high. As a result, simulations are conducted on grids that are selectively refined along the coastlines in order to provide the required accuracy of the solution in such highly volatile regions. Therefore in the studies [3, 63, 65], the tsunami models are equipped to handle nested and unstructured grids. Similarly, in the case of hydrological models, multiple studies [16, 29, 36, 43, 57] have been dedicated to solving the PDE on unstructured or non-uniform, structured grids. However, the drawback of a static, non-uniform grid is its ability to capture the wave-front of the water accurately. In dam break cases, it is essential to accurately predict the dry-wet interface which is time-dependent. Furthermore, in the majority of flooding scenarios taking place in urban landscapes, it is often hard to predict areas in the domain that requires higher resolutions.

Another solution to the challenges presented above would be to adapt the grid with time to obtain better accuracy in specific areas of the domain. By doing this, a comfortable level of autonomy is reached by delegating the task of refining the grid to the conditions provided by the scenario. Thus, it would result in lowering the error in the solution while increasing the conditioning and the efficiency of the model. Although, the approach of refining the mesh multiple times poses an additional memory overhead, the accuracy and the efficiency of the model are expected to outweigh the drawbacks. Some of the popular grid adaption techniques are presented below:

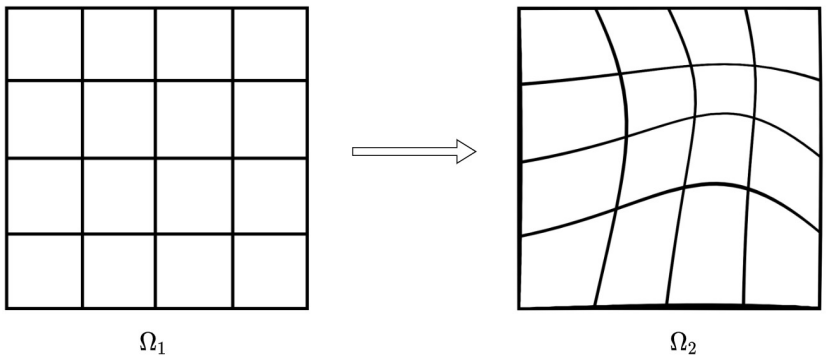



Figure 1.2: Illustration of a r-Refinement strategy

1. r-refinement (Relocation Refinement): Often called the moving mesh method [30, 61], it is a newer development in the area of mesh adaptivity. In r-refinement methods, the quadrature of the mesh is moved according to solution criteria. However, the number of mesh points, and the topology of the mesh is kept constant. Budd [12] provides a comprehensive account of the algorithms associated with moving meshes. Due to the fixed parameters of the mesh, r-refinement methods offer easier discretisation strategies in FEM. As the overall connectivity and topology do not change, hanging nodes (nodes on faces with neighboring cells having dissimilar refinement levels) are naturally eliminated. In the study by McRae [49], the general-purpose algorithm provided for generating moving meshes for non-linear PDE's offer these benefits. Furthermore, mathematical and computational overheads associated with modifying the load balancing and data structures are no longer an issue due to the fixed arrangement of the mesh points. Nevertheless, the moving nature of the mesh requires solving auxiliary PDE's to obtain the position and velocity of the mesh points. Thus, this adds to the cost of implementing a relatively novel method while bearing the computational expenses created by the additional degrees of freedom.

2. p-refinement: Similar to r-refinement methods, this approach is well suited to FEM. It involves non-uniform discretisation of PDE's across the domain. Based on a posteriori estimate computed using the solution, the local order of the polynomials is modified according to the threshold conditions. In one study [62], p-refinement strategies were used to solve the wave-continuity formulation of the shallow water equations. The method provided an accurate estimate of the surface elevation by maintaining continuity. Furthermore, it was demonstrated that when compared to lower order polynomials, quadratic elements produce more accurate results [5, 6]. Nevertheless,

this adaption strategy is suitable for parts of the domain where the solution is relatively smooth. Kubatko [37] uses a hybrid hp-refinement technique to tackle both smooth and discontinuous solutions.

3. h-refinement: Being one of the more widely researched approaches to grid adaptivity, it is employed by a number of commercial software suites. The h-refinement strategies involve refining and coarsening static grids for every time step based on a posteriori error estimate. It is shown to accurately capture highly discontinuous solutions and thus well suited to be used with FVM schemes [19, 34, 46, 48]. Numerous studies [11, 17, 18, 42, 46] have employed this methodology for geophysical phenomena such as storm surges, weather prediction, and dam breaks. However, using an improper selection of a posteriori error estimate or a lower order interpolation schemes may lead to non-physical numerical artifacts in the solution. Nevertheless, by using a balanced interpolation scheme in combination with a well-balanced numerical scheme, highly accurate solutions have been produced [13, 31, 50, 56]. Due to the proven background and benefits associated with FVM, the model created in the current project is based on the h-refinement scheme.

## 1.4. Open-Source Scientific Libraries

In light of developing an open-source model, a survey of the available open-source scientific libraries was conducted. Based on the method being used a variety of Computational Fluid Dynamics (CFD) codes were available. FEniCS [2] is one such platform for models based on the finite element method. It offers a python interface which accommodates for an easier translation of mathematical operators into the program. Furthermore, algorithms for adaptive mesh refinement can be instituted using the library as was done in the study [33].

An alternative for the finite volume method is offered by OpenFOAM [28]. With an extensive user-base and an active community, it features a comprehensive list of capabilities to develop hydrodynamic models. Numerous studies have been dedicated to exploring the h-refinement methods on oct-trees in 3-dimensional space and quad-trees in 2-dimensional space [15, 47, 51]. Nevertheless, OpenFOAM does not offer access to control every parameter associated with the numerical methodology.

PETSc [1, 7, 9, 64] is another counterpart to the high-level open-source scientific library. It provides a number of versatile modules and data structures to facilitate the production of massively parallel, large scale models for partial differential equations. Its solvers and meshing tools are used by other libraries such as OpenFOAM and FEniCS. With a large and active development community, a number of geophysical models were developed using the TS (Time-stepping) and ODE (Ordinary differential equation) solvers of PETSc [23, 52, 58]. As opposed to the other choices, PETSc can be considered as a library-oriented more towards the mathematical framework rather than the final application. Furthermore, with considerations of the functional requirements posed by SIM-CI (see section 1.2), it was decided to implement the model using PETSc.

## 1.5. Open-Source Meshing Tools

In favour of rapidly producing prototypes and trying to stay focused on testing and improving the existing h-refinement adaptive grid solutions for SWE, a decision was made to make use of existing open-source tools. By doing this, the time required for the development of the software is bypassed and the need to reinvent the wheel is thus eliminated. Therefore, a survey of available open-source tools was conducted and the results are discussed hereon.

Within the h-refinement approach, there exists a variety of strategies to increase the resolution of the grid. This could either be block or cell-based adaptive mesh refinement (AMR). In the block-based approach [22], the grid connectivity associated with the data structure remains small. On account of reduced changes in the grid resolutions, this approach has a smaller effect on creating numerical artifacts.

On the other hand, cell-based approach are focused on uniformly subdividing individual cells (parent) into smaller constituents (children). The algorithm, thus, relies more on tree structures to handle the resulting grid connectivity. Although it raises concerns in the implementation of the numerical method, it offers an easier route for load balancing among multiple processors. Therefore, highly scalable algorithms can be produced using this approach. Additionally, due to an increase in the number of ”blocks”

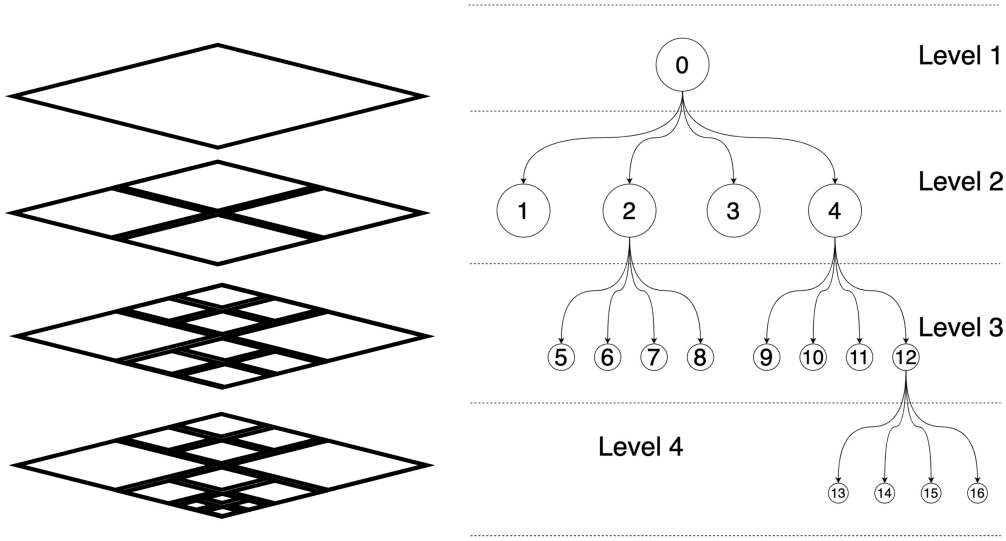



Figure 1.3: Cell-Based refinement strategy

in the grid, it can efficiently adapt to highly complex geometries.

### 1.5.1. Gmsh

Gmsh [24] is a popular open-source unstructured mesh generator that comes with pre- and post-processing utilities. It provides an attractive alternative to the other mesh generators as it offered a well-documented SDK. Furthermore, it has extensive features corresponding to adapting the h-refinement strategy [38]. Nevertheless, under the General Public License (GPL) of the software, it was prohibited to integrate into a closed-source software.

### 1.5.2. P4est

Another popular library for handling hierarchical quad- and oct-trees is p4est [14, 20]. It offers parallel cell-based AMR algorithms using linear storage. Using tree-based methods and schemes that are recursive, it does not allow overlapping refinement. Therefore, these attributes make it very efficient in storing distributed meshes and balancing loads.

### 1.5.3. DMForest

DMPLEX is a subset of methods and routines found within the DM module of PETSc [39]. It supports the storage and handling of various 3rd party mesh formats such as MED, Gmsh, and CGNS to name a few. The grid connectivity is then stored in the form of a directed acyclic graph (DAG). Each layer in the DAG represents a quadrature element of the grid. However, DMPLEX by definition was created to store and handle conformal meshes. The support for non-conformal, hierarchical meshes later came in the form of an interface called DMForest [32]. DMForest, essentially uses the algorithms provided by P4est and outputs the data in the unstructured grid format used by DMPLEX. Since DMPLEX offers the ideal platform to create models that call on the DM API of PETSc, it was chosen as the meshing tool for the subsequent models presented in this project.

## 1.6. Objectives

The need to model shallow water flows over the urban landscape is evident and its application exists for a number of governmental organisations. Furthermore, the necessity is heavily justified by the lack of well-documented, validated open-source hydrological solvers that support varying topography and adaptive mesh refinement. The objective of the project was thus to create a scalable, and efficient hydrological solver that features the h-refinement strategy. Furthermore, the hydrological solver must be capable of handling highly discontinuous initial conditions that are often encountered in real life

flooding scenarios.

With the decision to base the model on an open-source scientific library, a portion of the effort was dedicated to understanding and delineating the subsystems and subroutines offered by the library. Although the package is provided with a number of exercises, one might find the learning curve to be relatively steep. Therefore, it was recognized that a secondary objective of the project was also to provide a smoother ramp to understanding and effectively applying PETSc for the inception of other mathematical models. Hence, together with the comments provided in the source code, this report can be used as a building block for constructing other time-dependent numerical models based on PETSc.

# 2

# Problem Description

## 2.1. Scalar Advection Equation

In order to facilitate the understanding of the framework of PETSc, the strategy of divide and conquer was used. The bigger problem of solving a non-linear system of PDE's was broken into smaller parts. With simpler and smaller problems at hand, individual functionalities of a complex scientific library such as PETSc could be focused, studied and tested. In mathematical terms, the easiest route would be to create a functional solver for a scalar, linear PDE's and then move on to more complex PDE's such as the shallow water equations.

Being a scalar and a linear PDE, the creation and manipulation of the advection model was relatively simpler than that of the shallow water equations. The advection equation that is solved by the Riemann solver is given by:

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} + b\frac{\partial u}{\partial y} = 0 \tag{2.1}$$

Here, $a$ and $b$ is a constant. Now, consider an initial condition (Cauchy problem) given by

$$u(x, y, t = 0) = u_0(x, y)$$

We can derive the solution of the PDE to the initial condition as the following,

$$u(x, y, t) = u_0(x - at, y - at)$$

When $u(x, y, t)$ is plotted, we see that it moves along space and time without any deformations (characteristic lines in a x-t plane). Hence for this case, we have the exact solution for the initial problem and thus allowing a detailed error analysis and grid sensitivity analysis to be performed. The errors computed are norms of the differences between the exact solution and the numerical solution.

## 2.2. Shallow Water Equations

In terms of mathematical complexity, the next step would be to solve for a system of non-linear PDE. The two dimensional shallow water equations (SWE) exhibit these characteristics. The SWE is a deduction of the Navier-Stokes equations with the assumption of a hydrostratic condition for pressure. Besides, the velocities are assumed to be constant across the column of water. This essentially is obtained by neglecting the viscous effects. Due to relatively lower water column heights, the Coriolis effect is considered negligible as opposed to being a factor in tsunami modelling.

When considering the PDE without the involvement of a topography or the effects of friction, one obtains the SWE without any source terms. The first wave of models were modelled using SWE without bathymetry terms as a step to both expedite the process and test the efficacy of the AMR algorithm. The formulation of this system is shown in eq. 2.2.

$$\frac{\partial \mathcal{U}}{\partial t} + \frac{\partial \mathcal{F}}{\partial x} + \frac{\partial \mathcal{G}}{\partial y} = 0 \tag{2.2}$$

By taking into account the effect of frictional terms ($S_f$) and topographic slopes ($S_b$), the SWE can be expanded as

$$\frac{\partial \mathcal{U}}{\partial t} + \frac{\partial \mathcal{F}}{\partial x} + \frac{\partial \mathcal{G}}{\partial y} = S_b(\mathcal{U}) + S_f(\mathcal{U}) \tag{2.3}$$

Here, the vectors are defined as the following:

$$\mathcal{U} = \begin{bmatrix} h \\ uh \\ vh \end{bmatrix} \quad \mathcal{F} = \begin{bmatrix} uh \\ u^2h + \frac{1}{2}gh^2 \\ uvh \end{bmatrix} \quad \mathcal{G} = \begin{bmatrix} vh \\ uvh \\ v^2h + \frac{1}{2}gh^2 \end{bmatrix}$$

$$\mathcal{S}_b = \begin{bmatrix} 0 \\ -gh\frac{\partial z_b(x,y)}{\partial x} \\ -gh\frac{\partial z_b(x,y)}{\partial y} \end{bmatrix} \quad \mathcal{S}_f = \begin{bmatrix} 0 \\ -\frac{\tau_x(u,v)}{\rho} \\ -\frac{\tau_y(u,v)}{\rho} \end{bmatrix}$$

$$\tau_x(u,v) = \rho C_f u\sqrt{u^2+v^2} \tag{2.4}$$

$$\tau_y(u,v) = \rho C_f v\sqrt{u^2+v^2} \tag{2.5}$$

where $z_b(x,y)$ is the bed elevation of the domain. As shown in eq. (2.4) and (2.5), $\tau_x(u,v)$ and $\tau_y(u,v)$ account for the friction offered by the bed. They are also a function of the bed roughness coefficient which is given by

$$C_f = g\frac{n^2}{h^{1/3}}$$

Furthermore, n is called as the manning coefficient. It is estimated through empirical methods that are carried out through controlled measurements of the flow of liquids over different terrains.

# 3

# Numerical Methodology

## 3.1. Scalar Advection Equation

For the scalar advection equation, the discretisation was done using the finite difference method. This choice is motivated by the use of the DMDA module of PETSc. Suited to handle uniform meshes, the DMDA module offers routines store and handle adjacency information of the respective cell centers. This naturally results in a straight forward procedure to implement the finite difference method.

Consider eq.(2.1), the spacial derivatives are discretized using the first order upwind method. Suppose for a inner cell $(i, j)$, once the spatial discretization is carried out, the equation then becomes

$$\frac{\partial u(x,y,t)}{\partial t} = -[a^+ u_x^- + a^- u_x^+] - [b^+ u_y^- + b^- u_y^+] \tag{3.1}$$

where

$$\begin{aligned} a^+ &= max(a, 0), \quad a^- = min(a, 0) \\ u_x^- &= \frac{u_{i,j}^n - u_{i-1,j}^n}{\Delta x}, \quad u_x^+ = \frac{u_{i+1,j}^n - u_{i,j}^n}{\Delta x} \\ b^+ &= max(b, 0), \quad b^- = min(b, 0) \\ u_y^- &= \frac{u_{i,j}^n - u_{j-1}^n}{\Delta y}, \quad u_y^+ = \frac{u_{i,j+1}^n - u_{i,j}^n}{\Delta y} \end{aligned}$$

For the purpose of verifying the solver's numerical order of accuracy, the Forward Euler method is used to discretise in the temporal domain. Eq.(3.1) thus becomes

$$u_{i,j}^{n+1} = u_{i,j}^n - \Delta t \left( [a^+ u_x^- + a^- u_x^+] + [b^+ u_y^- + b^- u_y^+] \right)$$

## 3.2. Shallow Water Equation

The spatial discretisation of the two dimensional shallow water equations are done using the finite volume based Godunov type approach.

Let us reconsider eq.(2.2) and write it in the form of a differential conservation law,

$$\mathcal{U}_t + \mathcal{F}(\mathcal{U})_x + \mathcal{G}(\mathcal{U})_y = 0 \tag{3.2}$$

Now, the integral form can be written as:

$$\frac{d}{dt} \int \int_{\mathcal{V}} \mathcal{U} d\mathcal{V} + \int_{\Omega} \mathcal{H}.\hat{n} d\Omega = 0$$

where $\Omega$ is the domain boundary, $\mathcal{V}$ represents the control volume, $\mathcal{H} = [F, G]^T$ corresponds to the flux and $\hat{n}$ is the normal vector in the respective Cartesian coordinates.

For two dimensions, the cell average of conserved variables across the cell *area* can be computed using

$$\hat{\mathcal{U}} = \frac{d}{dt}\int\int_V \mathcal{U} dV \tag{3.3}$$

Furthermore, the average of the net sum of normal fluxes through the interfaces of the cell can be computed from the integral.

$$\frac{1}{|A|}\sum_{p=1}^{N}\mathcal{F}_p = \frac{1}{|A|}\int_\Omega \mathcal{H}.\hat{n}d\Omega \tag{3.4}$$

Using eq.(3.3) and (3.4), the semi-discrete form can be written where the flux term is discretised in space while the cell average is left continuous in time,

$$\frac{d}{dt}\hat{\mathcal{U}} + \frac{1}{|A|}\sum_{p=1}^{N}\mathcal{F}_p = 0 \tag{3.5}$$

where $\mathcal{F}_p = \int_{A_p}^{A_{p+1}}[n_1\mathcal{F}(\mathcal{U}) + n_2\mathcal{G}(\mathcal{U})]d\Omega$. For a two dimensional space composed of $I_{(i,j)}, \forall i,j \in [1,n]$, where every cell is a quadrilateral, and the appropriate cell widths in the x and y directions are $\Delta x$ and $\Delta x$ respectively, eq.(3.5) can be written as

$$\frac{d}{dt}\hat{\mathcal{U}} = -\frac{1}{\Delta x}(\mathcal{F}_{i+\frac{1}{2},j} - \mathcal{F}_{i-\frac{1}{2},j}) - \frac{1}{\Delta y}(\mathcal{G}_{i,j+\frac{1}{2}} - \mathcal{G}_{i,j-\frac{1}{2}}) \tag{3.6}$$

The numerical method is considered complete once the flux terms at the interfaces, i.e., $\mathcal{F}_{i+\frac{1}{2},j}$, $\mathcal{G}_{i,j-\frac{1}{2}}$, are defined. The choice of these terms are further decided by the numerical scheme picked. The Godunov flux is picked as the choice of the flux to provide a complete and a conservative scheme.

## 3.3. Godunov Upwind Method

Assuming that we have a solution which when distributed across a finite grid, is decomposed into n piece-wise constant solutions. The intuition behind the Godunov method is that each of these piece-wise constant solutions are travelling at a finite speed and can be computed. Thereafter, the final solution can be exactly computed by solving the Riemann problem at the interfaces of each cell. Without any extensions, the scheme is first order accurate. For brevity and clarity, the Godunov method is represented by the one dimensional version of the shallow water problem.

$$\mathcal{U}_t + \mathcal{F}(\mathcal{U})_x = 0 \tag{3.7}$$

Following the procedure in section 3.2, eq.(3.7) can be discretized in space and time and be represented as the following

$$\mathcal{U}_i^n = \mathcal{U}_i^{n+1} - \frac{\Delta t}{\Delta x}(\mathcal{F}_{i+\frac{1}{2}} - \mathcal{F}_{i-\frac{1}{2}}) \tag{3.8}$$

Now consider $x = i + \frac{1}{2}$ in a tiny fraction of the domain, $x_{i-\frac{3}{2}} \le x \le x_{i+\frac{1}{2}}$. Here we can see that it can be reduced to a initial value problem (IVP) which consists of a discontinuity at $x = i + \frac{1}{2}$ and so on. Considering the same PDE governing the problem, the initial value of $\mathcal{U}$ for the specified domain can be written as

$$\mathcal{U}(x,t_n) = \begin{cases} \mathcal{U}_i^n, & \text{if } x < x_{i+\frac{1}{2}} \\ \mathcal{U}_{i+1}^n, & x > x_{i+\frac{1}{2}} \end{cases}$$

Using the same logic, a similar IVP can be instituted for the interface at $x_{i-\frac{1}{2}}$. In order to compute the flux at the interface, say $(\mathcal{F}(\mathcal{U})_{i+\frac{1}{2}})$, where the value of $\mathcal{U}_{i+\frac{1}{2}}$ is such that along the t axis, we have $x/t = 0$. Therefore, to find the value of $\mathcal{U}_{i+\frac{1}{2}}(x/t = 0)$, one needs to solve the Riemann problem corresponding to eq.(3.7). The exact choice and type of Riemann solver is explained further in the upcoming section.

## 3.4. Riemann Solver

Reconsidering eq.(3.8), we assume that the flux terms $\mathcal{F}_{i+\frac{1}{2}}$ and $\mathcal{F}_{i-\frac{1}{2}}$ do not change over time. Then, the fluxes can be exactly computed using the values provided by the solution of the Riemann problem. Depending on the type of case (rarefaction wave or a shock), the Riemann solver provides the values at the interface. Consequently, one can compute the flux values using those values.

Depending on the complexity of the equation, solving the exact Riemann fluxes at the interface can be very computationally expensive. For the two dimensional scalar advection equation, the use of an exact Riemann solver is very much feasible. This is because one has to only worry about a single wave propagating at every interface due to the simplicity of the equation. However, it is a different case when it comes to the shallow water equations where multiple waves propagating at the different speeds and possibly in different directions. As a consequence, utilization of an exact Riemann solver becomes impractical.

The exact Riemann solver in this case can be considered to be redundant as there is not a significant gain in accuracy when shifting from exact to approximate Riemann solvers. Furthermore, approximate Riemann solvers are not as computationally expensive when compared to their counterparts. Therefore in the following sections, the approximate Riemann solvers used in shallow water equations are described.

### 3.4.1. Rusanov Approximate Riemann Solver

We know that when a naive average of the flux in the left and side of the interface is considered, the method is unconditionally unstable for hyperbolic systems. However the instabilities can be eliminated by adding enough viscosity to the average. The magnitude of the stability can be predicted by the largest local wave speed given by the eigenvalues of $\mathcal{A}$. The largest of them can be considered as the viscosity coefficient. As an illustration, for a hyperbolic system, one has

$$\frac{\partial \mathcal{U}}{\partial t} + \frac{\partial \mathcal{F}(\mathcal{U})}{\partial x} = 0$$

Then the Rusanov Flux is defined as the following:

$$\mathcal{F}_{rusanov}(\mathcal{U}_L - \mathcal{U}_R) = \frac{1}{2}(\mathcal{F}(\mathcal{U}_L) + \mathcal{F}(\mathcal{U}_R) - \lambda_{max}(\mathcal{U}_L - \mathcal{U}_R)) \tag{3.9}$$

Here, $u_L$ and $u_R$ are values of the state vectors based on the left and right cells of the corresponding interface.

In the case of the shallow water equations, there are 3 eigenvalues for this system. The maximum wave speed is thus given by:

$$\lambda_{max} = \max_{\mathcal{U}\in[\mathcal{U}_L - \mathcal{U}_R]} |\lambda(\mathcal{F}'(\mathcal{U})| = max(u^L + \sqrt{gh_L}, u^R + \sqrt{gh_R})$$

Here, $n_x$ and $n_y$ are normals in the x and y direction. However, it must be noted that the choice of the wave speed is restricted. For stability, the following condition must be satisfied

$$\lambda_{max} \leq \frac{\Delta x}{\Delta t} \tag{3.10}$$

In the case of an equality in eq.(3.10), then the flux provided by eq.(3.9) results in a Lax Friedrichs flux. To establish a stable condition for each timestep, an estimate of the maximum wave speed must be known. Using the combination of the wave speed and the Courant-Friedrichs-Lewy value (CFL) $C$, one can compute a maximum timestep value.

$$\Delta t = C\frac{\Delta x}{\lambda_{max}}$$

### 3.4.2. HLL Approximate Riemann Solver

Similar to the Rusanov approximate Riemann solver, this solver takes into account an estimate of the minimum and the maximum wave speeds. In this approach, the effects of sheer waves and any intermediate waves are assumed to be negligible. As shown in figure 3.1, let us again consider the

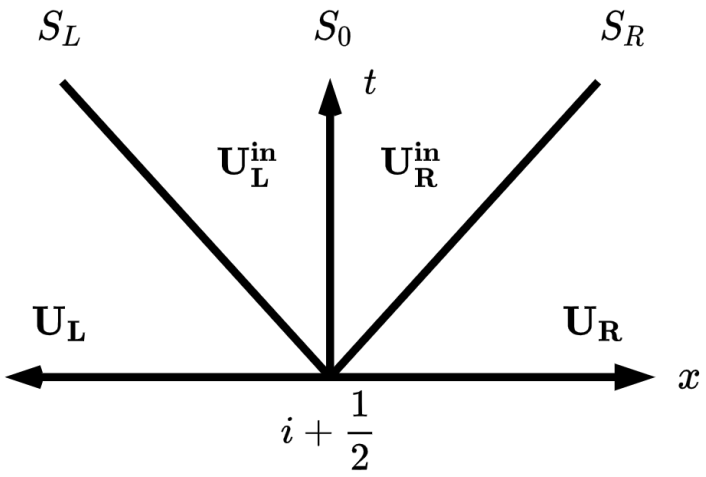



Figure 3.1: Wave structure of the Riemann problem at the interface $x = i + \frac{1}{2}$

interface $i + \frac{1}{2}$ where we set the state vectors $\mathcal{U}_L \equiv \mathcal{U}_i$ and $\mathcal{U}_R \equiv \mathcal{U}_{i+1}$ as the left and right states of the interface. Similarly for flux vectors we set $\mathcal{F}_L \equiv \mathcal{F}(\mathcal{U}_i)$ and $\mathcal{F}_R \equiv \mathcal{F}(\mathcal{U}_{i+1})$.

Then the flux at the interface is decided based on the estimated speeds of the left-going and right-going waves. A number of different formulations for the estimates of the wave speeds are available. This choice is again made to obtain the best resulting accuracy and stability for the scheme.

$$\mathcal{F}_{i+\frac{1}{2}} = \begin{cases} \mathcal{F}_L, & \text{if } S_L > 0 \\ \mathcal{F}_{riem}, & \text{if } S_L \leq 0 \geq S_R \\ \mathcal{F}_R, & \text{if } S_R \leq 0 \end{cases}$$

The HLL flux is given by,

$$\mathcal{F}_{riem} = \frac{S_R \mathcal{F}_L - S_L \mathcal{F}_R + S_R S_L (\mathcal{U}_R - \mathcal{U}_L)}{S_R - S_L} \tag{3.11}$$

As mentioned before, the wave speeds are chosen such that it handles the interfaces of dry neighbors. Therefore, an appropriate condition is necessary for the left and right wave speeds, and the intermediate parameters as described by Toro [54].

$$S_L = \begin{cases} \hat{u}_R - 2\sqrt{g h_R}, & \text{if } h_L = 0 \\ min(\hat{u}_L - \sqrt{g h_L},\, u_{in} - \sqrt{g h_{in}}), & \text{if } h_L > 0 \end{cases}$$

$$S_R = \begin{cases} \hat{u}_L + 2\sqrt{g h_L}, & \text{if } h_R = 0 \\ min(\hat{u}_R - \sqrt{g h_R},\, u_{in} + \sqrt{g h_{in}}), & \text{if } h_R > 0 \end{cases}$$

where, $a_{L,R} = \sqrt{g h_{L,R}}$ is the gravity wave speed and $\hat{u}_{L,R} = u_{L,R}\, n_x + v_{L,R}\, n_y$ is the normal velocity with respect to the interface. Furthermore, $u_{L,R}$ and $v_{L,R}$ are the velocities in the x and y coordinates with respect to the interface. The intermediate variables $u_{in}$ and $h_{in}$ are given by

$$h_{in} = \frac{1}{g}\left(\frac{1}{2}(\sqrt{g h_L} + \sqrt{g h_R} + \frac{1}{4}(\hat{u}_L - \hat{u}_R)\right)^2 \tag{3.12}$$

$$u_{in} = \frac{1}{2}\,(\hat{u_L} + \hat{u_R}) + \sqrt{g h_L} - \sqrt{g h_R} \tag{3.13}$$

### 3.4.3. Augmented Riemann Solver

The topography in the domain can be taken into consideration with the addition of the source term containing the bed slope term. Rewriting eq.(2.3) without the source term associated with friction, we have

$$\frac{\partial \mathcal{U}}{\partial t} + \frac{\partial \mathcal{F}}{\partial x} + \frac{\partial \mathcal{G}}{\partial x} = S_b(\mathcal{U})$$

Let the topography $z(x, y)$ be a function of the domain. Here the source term associated with the topography is defined as,

$$\mathcal{S}_b(\mathcal{U}) = \begin{bmatrix} 0 \\ -gh\frac{\partial z(x,y)}{\partial x} \\ -gh\frac{\partial z(x,y)}{\partial y} \end{bmatrix}$$

For clarity and simplicity, the formulation of the augmented HLL Riemann solver provided by Audesse [4] is only provided for the interfaces with normals in the x direction. One can easily derive the corresponding formulations in the y direction using the same procedure.

$$\lim_{\mathcal{U}_L, \mathcal{U}_R \to \mathcal{U} \text{ as } \Delta x \to 0} \mathcal{S}_b(\mathcal{U}_L, \mathcal{U}_R, z_L, z_R) = \begin{bmatrix} 0 \\ -gh\frac{\partial z(x,y)}{\partial x} \end{bmatrix} \tag{3.14}$$

With the approximation shown in eq.(3.14), they can be transferred to the LHS and grouped into the fluxes. Furthermore, one of the primary motivations of reformulating the flux terms using the bathymetry term is to provide a well balanced model that are consistent in steady state cases. One of such cases involve a level water surface height with a varying topography. In the absence of a balancing term in the flux formulation, there will be non-physical fluxes resulting in a scheme that does violates conservation of mass. Thus, for states $\mathcal{U}_s$ and $\mathcal{U}_t$, a well balanced scheme should satisfy the hydraulic balance condition given by

$$\mathcal{U}_s^1 + b_s = \mathcal{U}_t^1 + b_t \tag{3.15}$$

$$\mathcal{U}_s^2 = \mathcal{U}_t^2 \tag{3.16}$$

From eq. (3.12) and (3.13), we can see that the intermediary states $u_{in}$ and $h_{in}$ will not depend on the bathymetry terms. This in turn implies that the fluxes left and right of the interface at $x = i + \frac{1}{2}$, can also be defined as a function of the bathymetry $z_L$ and $z_R$ respectively. They are given by

$$\mathcal{F}^L_{i+\frac{1}{2}}(\mathcal{U}_L, \mathcal{U}_R, z_L, z_R) = \mathcal{F}_{riem} + \begin{bmatrix} \frac{S_L S_R (z_R - z_L)}{S_R - S_L} \\ \frac{-S_L g \Delta x h \frac{\partial z}{\partial x}}{S_R - S_L} \end{bmatrix} \tag{3.17}$$

$$\mathcal{F}^R_{i+\frac{1}{2}}(\mathcal{U}_L, \mathcal{U}_R, z_L, z_R) = \mathcal{F}_{riem} + \begin{bmatrix} \frac{S_L S_R (z_R - z_L)}{S_R - S_L} \\ \frac{-S_R g \Delta x h \frac{\partial z}{\partial x}}{S_R - S_L} \end{bmatrix} \tag{3.18}$$

where $\mathcal{F}_{riem}$ is given by eq.(3.11). Furthermore, the well-balanced property is provided by adopting a natural discretisation for the bed slope terms. It is given by

$$h\frac{\partial z}{\partial x} = \frac{1}{\Delta x}\frac{h_L + h_R}{2}(z_R - z_L)$$

## 3.5. MUSCL-Hancock Scheme

The PETSc TS module allows the configuration of various time stepping methods such as the Euler Forward Method, Euler Backward method, Generalised Linear Methods, Runge-Kutta Time stepping schemes to name a few. An explicit time stepping method is chosen over an implicit one as it was found to be far too computationally expensive. When coupled with time-adaptive meshes, the advantages offered by implicit time-stepping methods is outweighed by its higher FLOP (Floating point operations) count and memory usage. To achieve an overall second-order accurate scheme, the MUSL-Hancock

method is chosen. According to the scheme, the solution is updated in two steps, namely the predictor and the corrector step. In the first step, the solution is marched forth by half a time step. Expanding on eq.(3.6) and dropping the ˆ symbol, we have

$$\mathcal{U}_{i,j}^{n+\frac{1}{2}} = \mathcal{U}_{i,j}^{n} + \frac{\Delta t}{2\Delta x}(\mathcal{F}_{i+\frac{1}{2},j}^{n} - \mathcal{F}_{i-\frac{1}{2},j}^{n}) + \frac{\Delta t}{2\Delta y}(\mathcal{G}_{i,j+\frac{1}{2}}^{n} - \mathcal{G}_{i,j-\frac{1}{2}}^{n})$$

Here the flux terms are computed using the appropriate Riemann solver. However, the state vectors required by the Riemann solver are linearly interpolated values from the cell centers to the corresponding face centroids. To suppress spurious oscillations, various limiters such as of types Minmod, Van-Leer, or Superbee can be used in the reconstruction procedure. For instance, such an interpolation for the state vector at a position $(x, y)$ based on a cell centered value at $(x_i, y_j)$ is given by

$$\mathcal{U}(x, y) = \mathcal{U}_{i,j} + r \,.\nabla \mathcal{U}_{i,j}$$

where r is the distance vector from the cell center to point under consideration and $\nabla \mathcal{U}_{i,j}$ is the gradient of the state vector. Next the corrector step follows, but now the reconstruction of the state vector is done based on the updated values i.e., $\hat{\mathcal{U}}^{n+\frac{1}{2}}$. This step is thus given by,

$$\mathcal{U}_{i,j}^{n+1} = \mathcal{U}_{i,j}^{n+\frac{1}{2}} + \frac{\Delta t}{2\Delta x}(\mathcal{F}_{i+\frac{1}{2},j}^{n+\frac{1}{2}} - \mathcal{F}_{i-\frac{1}{2},j}^{n+\frac{1}{2}}) + \frac{\Delta t}{2\Delta y}(\mathcal{G}_{i,j+\frac{1}{2}}^{n+\frac{1}{2}} - \mathcal{G}_{i,j-\frac{1}{2}}^{n+\frac{1}{2}})$$

The fluxes at the interface must be computed twice. Thus, the Riemann solver is utilized twice for every time step marched forward. As a result, the `TSSetRHSFunction()` is called twice for every time step.

## 3.6. Stability Criteria

As an explicit scheme is used, the time step value is limited by the maximum wave speeds in the two coordinates, $S_x$ and $S_y$. For the two models they are specified below for $\Omega = (i, j) \in \{(1, m) \times (1, n)\}$:

- Scalar Advection Equation : $S_x = u_{i,j}$ and $S_y = v_{i,j}$
- Shallow Water Equations : $S_x = u_{i,j} + \sqrt{g h_{i,j}}$ and $S_y = v_{i,j} + \sqrt{g h_{i,j}}$

This criteria is measured by the CFL number. The time step is thus computed from the following,

$$\Delta t = \mathrm{C} \min_{\Omega} \left( \min_{\Omega} \frac{(\Delta x)_{i,j}}{S_x} , \min_{\Omega} \frac{(\Delta y)_{i,j}}{S_y} \right)$$

For the subsequent test cases illustrated, a CFL number of 1 has been used. Note that if the implicit time stepping solver is used, the value of the time step is no longer bounded by the CFL number.

# 4

# Implementation

## 4.1. PETSc

Portable, Extensible Toolkit for Scientific Computation (PETSc) is a compilation of data types and routines developed for numerically modelling of partial differential equations. PETSc is suited to create highly scalable solvers which perform large scale scientific computations through the usage of MPI and/or GPU accelerators. It consists of different linear PDE, non-linear PDE, and ODE solvers. Most of the implementation of the library is carried out in C and C++ with additional wrappers written in Python.

The purpose of choosing PETSc to base the models presented in this report can be illustrated through several of its features. They are given below:

- Parallelization: PETSc offers an intuitive interface to work with parallel vectors and matrices. A majority of the underlying mechanisms needed for a parallel programming are hidden behind user-friendly interfaces and routines. These routines are relatively simple to conceptualize from the perspective of an applied mathematician. Operations such as communication of ghost points between domains, assembly of matrices are facilitated through the functions already available in the library.
- Modularity: PETSc is divided into several subsystems which are responsible for handling various functions in the subsequent numerical modeling of the mathematical model. They have been structured in a manner that promotes low coupling between subsystems and high coherence within a subsystem. More information on each of the subsystem is provided later in section 4.2.
- Post-processing Options: The library also provides a myriad of methods to process the output of the solution. The solution thus, can be converted and visualized in 3rd party applications such as MATLAB, HDF5Viewer or ParaView to name a few. The availability of multiple choices proved to be extremely valuable during the visualizations of unstructured, adaptive grid solutions.
- Comprehensive Debugging : PETSc can be configured to be used in conjunction with several open-source debuggers such as lldb and hrefhttps://valgrind.orgvalgrind. Thus, identifying bugs, memory leaks and running benchmarks for models is facilitated by this support.
- Software Support : There is an extremely active community working on improving and providing additional tutorials for the library. PETSc enjoys support and contribution from academic and industrial researchers based all across the globe. The primary developers provided valuable suggestions during the project which helped smoothen the learning curve and accelerated the progress.

## 4.2. System Decomposition

In figure 4.1, an overview of system decomposition is shown. As mentioned earlier, PETSc is a 'collection' of modules using the fundamental frameworks already set in place at the lowest level of abstraction. A brief description of each of the module is given below.

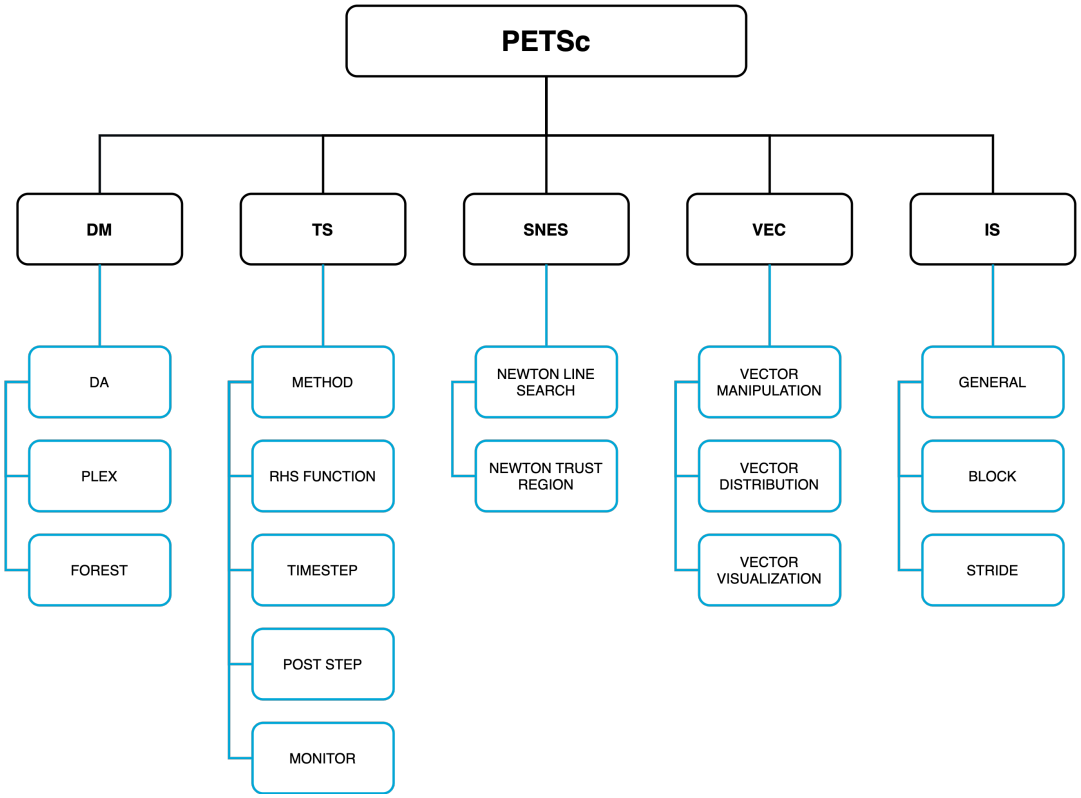



Figure 4.1: System Decomposition of PETSc

### 4.2.1. VEC

`VEC`, a data type defined in PETSc, is meant for storing, accessing and manipulating vectors associated with the model. It can be considered as one of the simplest type of a PETSc data type and is suitably located at the very bottom of the abstraction level. In the context of numerical modelling of partial differential equations (PDE), `VEC` is critical to a majority of the operations as solutions, quadrature information and adaptive grid parameters which are all stored in the form of vectors. Some of the basic routines and its description is given below:

- Vector Creation and Assembly: The `VEC` object is created by passing the MPI comm object and the initialized vector. Preliminary operations such as setting the size and values follow after the creation of the `VEC` itself. Once the object is defined, `VecAssemblyBegin()` and `VecAssemblyEnd()` has to be called to complete the assembly of the object.
- Vector Distribution: Some of the operations are dedicated to the management of the storage and distribution of the `VEC` object across multiple domains and/or processors. For example, the function `VecGetOwnershipRange()` allows the user to obtain the indices of the vector in range for a given processor. Different routines are available for 'Read-Only' or 'Read-Write' access of the vectors such as `VecGetArray()`. This must be followed by the routine `VecRestoreArray()` to record the changes.
- Vector Operations: Most of the basic operations such as basic arithmetic are provided for the `VEC` operations. Furthermore, Norm computations are facilitated with specific routines. They are automatically suited for both parallel and serial vectors. However, operations between a serial and a parallel vector are not allowed.

### 4.2.2. Index Sets (IS)

To implement the operations of scattering and gathering in Vector objects, the Index Set (IS) object is created. It essentially refers to the variable that contains the indices of the vectors or matrices. In the majority of the routines, the `IS` object is used to store the processor ranges of `VEC` objects. Additional

usages include the storage of indices of the inter-processor and boundary ghost cells. In the context of the adaptive grid method, the `IS` object is used to store the indices of the cells that are tagged to be refined or coarsened. This is explained in later sections.

### 4.2.3. SNES

This module provides a suite of routines to tackle large scale non-linear systems. Much of the methodologies are based on the linear solvers (KSP) which are not covered in this report. With an identical interface to other solvers, the data structures used by SNES are in turn similar. The nonlinear equations are solved from the form given below:

$$F(x) = 0$$

As provided in the PETSc manual, the solver uses Newton's method to solve the non-linear problem. Therefore, the solution vector of n-dimension is computed using the following equation:

$$x_{k+1} = x_k - J(x_k)^{-1} F(x_k)$$

where $k = 0, 1, ..., n$ The user thus, has to provide a routine to assemble the Jacobian matrix $J(x_k)$ and define the form function $F(x_k)$.

### 4.2.4. Time Stepping (TS)

With the fundamental elements of the framework (`VEC` and `IS`) in place, Ordinary Differential Equation (ODE) solvers and Differential Algebraic System (DAE) solvers can be solved using the TS module of PETSc. Regardless of the numerical method used, the TS module solves for the equation written in the form:

$$F(t, u, \dot{u}) = G(t, u) \tag{4.1}$$

Where the initial condition is defined by $u(t_0) = u_0$. Here in the context of PETSc programs, the term $G(t, u)$ is generally called as the right-hand side (RHS) function. There are options for either providing an user-defined routine for the RHS expression (typically in the case of cell-based finite difference method). Nevertheless, the models used in the project were based on finite volume methods. Therefore, the RHS expression was computed using residual evaluation routines predefined in the TS module.

While constructing a transient model using PETSc, it is essential to define a few essential parameters for the solver. They are described as follows:

- `TSSetType()` : Sets the type of time stepping solver to be used. The temporal discretisation can be chosen from a variety of methods prescribed in the TS module. Some of them include, forward and backward Euler, Runge-Kutta methods and, general linear methods. More of such options is given in Table 11 of [8].
- `TSSetTimeStep()` : Sets the timestep value at the start of every time iteration. The user is responsible in providing a routine to effectively compute the timestep $dt$ from the grid and solution parameters.
- `TSSetPostStep()` and `TSSetMonitor()` : These routines correspond to the post computation step of the solver. They are essentially used for monitoring the residuals and producing the output in various formats.

### 4.2.5. Data Management (DM)

As per the definition provided by PETSc [8], `DM` provides a collection of routines and data structures to define and facilitate the communication between the `VEC` and `MAT` objects and the distribution of mesh elements corresponding to a particular PDE. Apart from some basic functions, `DM` itself is divided into 3 major types.

- Distributed Arrays (DMDA): It is responsible for creating and handling Cartesian structured meshes. The interface for the topology and geometry is provided by this class of objects. Furthermore, it is also capable of parallel refining and coarsening. Routines such as `DMDACreate2D()` and

`DMDACreate3D()` can be used to create two and three dimensional uniform meshes respectively. Although highly optimised for the finite difference method, the drawback of DMDA lies in its lack of support for non-uniform, unstructured meshes.

- Plex: The void created by DMDA is filled in by the DM of Plex type. They are suited for handling of unstructured and/or non-conformal meshes. DMPlex is very well suited for problems using the finite volume or the finite element method as it allows the user to loop over the cell interfaces. Furthermore, the functionality of reading from a 3rd party mesh format and store it as `DM` objects are also available.

- Forest: It can be considered as a derivative of DMPlex. It is suited to handle parallel quad-trees in 2D and oct-trees in 3D. In the context of the final model using the adaptive grid method, the DMForest contains routines responsible for handling the mesh information and the subsequent interpolation associated with adapting the mesh from one refinement level to another.

## 4.3. Finite Volume Method Framework on PETSc

In this section and in figure 4.2, the implementation of the Finite Volume Method (FVM) in PETSc is described in detail. As an example, the advection model is chosen for this section. Nevertheless, the rudimentary flow of the process remains unchanged for other PDE's using FVM. The flow of the process is divided into 3 sections as listed below.

1. User Context: The user context is a custom `struct` defined by the user to store model or physics-related parameters. This would include, coefficients and constants whose access is needed by multiple modules down the line of the process. For example, CFL number, gravitational constant, or advection coefficient could be part of the user context. Some important PETSc data structures such as the `PetscFV` object can also be included in the user context as was done for the adaptive grid model.

2. DMPLEX: Being a `DM` object of type Plex, it contains all of the essential information corresponding to the mesh. Inputs such as custom labels for the type of boundary conditions, size, and type of the grids can be specified using the routines provided by the DM library. In all of the models used in the project, `DMPlexCreateBoxMesh()` is used to create the data structure for the mesh. Furthermore `DMSetAdjacency()` is used to configure the `DM` object to store adjacency information about various mesh elements such as cell centers and cell faces. The number of cells marked for overlap is also specified to allow the parallelisation of the problem. Subsequently, it is followed by the distribution of the ghost cell locations.

3. PETSc FV: Specific functions related to FVM are available through the `PetscFV` object. Here, the user has to provide the number of fields, degrees of freedom, and the dimension. In the case of an advection model, the number of fields and degree of freedom are equal to one. A similar translation could be made for the shallow water equations. Furthermore, the type of the `PetscFV` object decides the interpolation type (Least Squares or Upwind).

4. Discrete System (DS): In the process of creating a finite volume model, the spatial discretisation of the system is provided to the TS module through the `DS` object. The specification of the Riemann solver is provided through the `DS` object. Moreover, the routine used to compute the time derivative values for the TS module uses the data structure defined in the `DS` object.

5. Time Stepping (TS): The solution vector X is marched forward in time by the respective time stepping solver specified in the TS object. As mentioned before, using the `PetscFV` and `DS` object, $F(t, u, \dot{u})$ in eq. 4.1 is computed. With the appropriate time-step $dt$, the solution vector of the next timestep is computed followed by the routine for post-processing is executed. As a final step, the solution can be outputted into a file, for instance, in the *.vtu* format for visualization purposes.

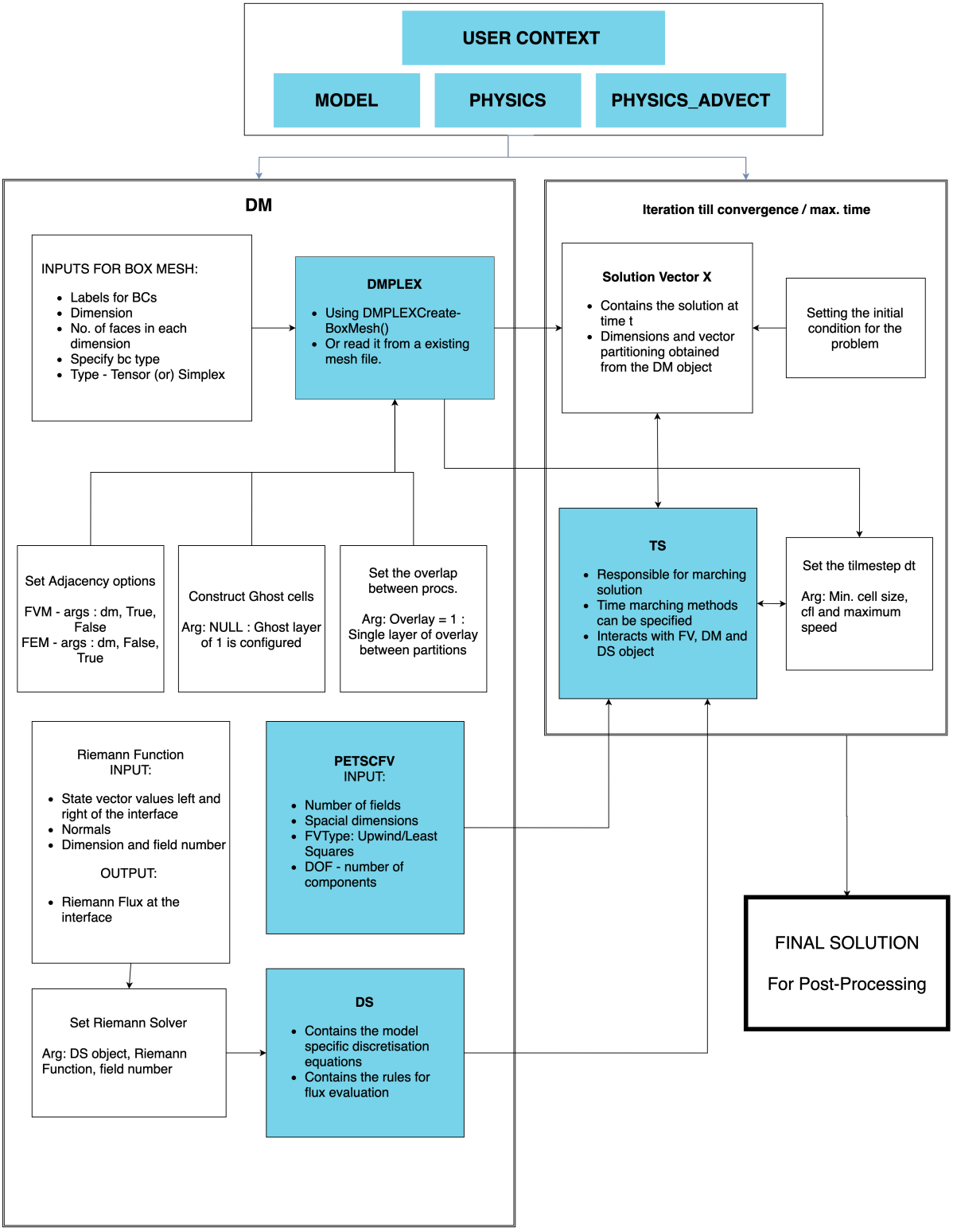



Figure 4.2: Process flow for a Finite Volume Method Model in PETSc

## 4.4. Routine to Compute Time Derivative

The steps involved in the course of the computation of time derivative in eq. 4.1 is described in detail hereon.

1. The initial step in the process remains the same as illustrated in the fig. 4.2. As mentioned in

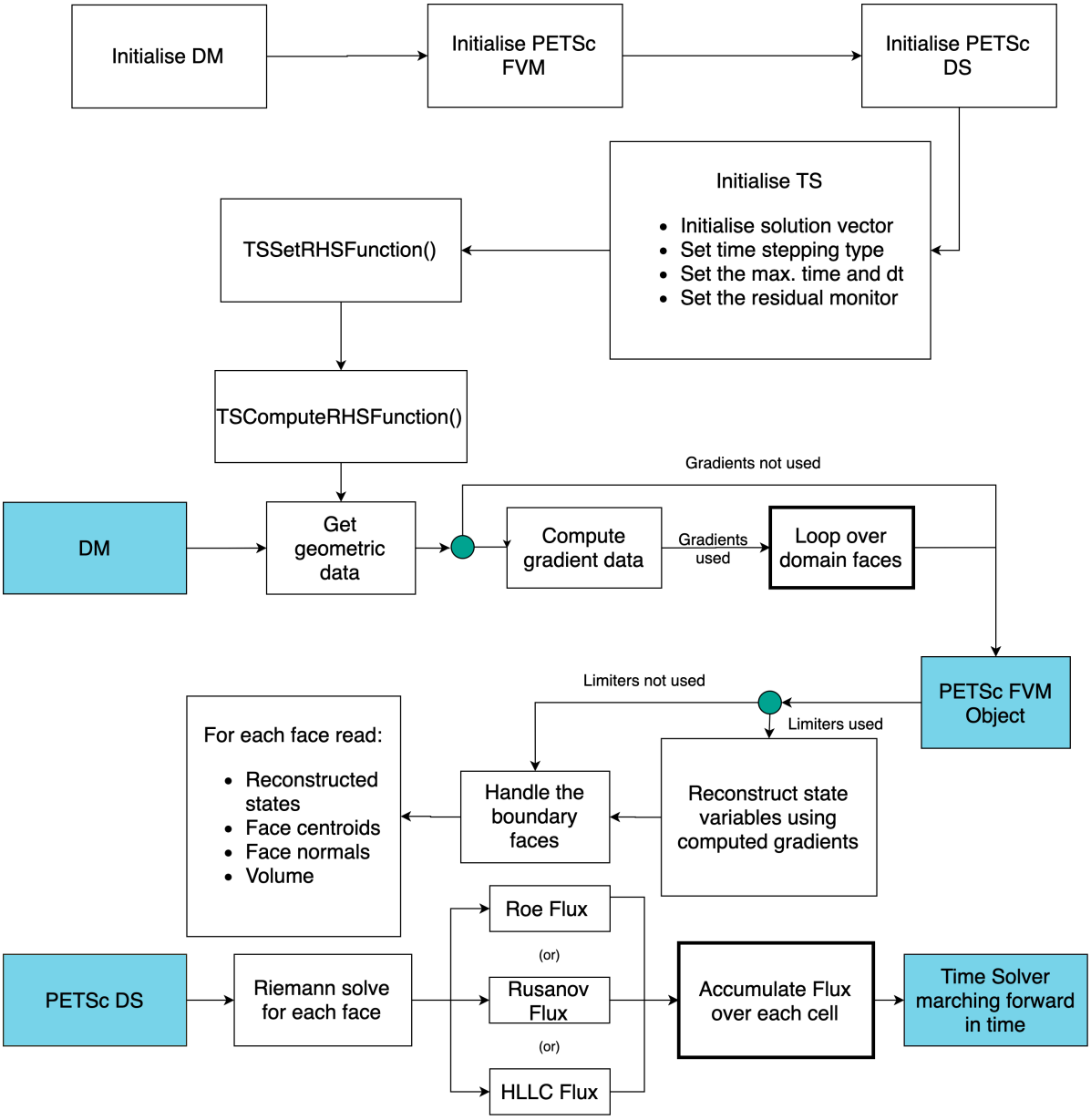



Figure 4.3: Computation of the Right Hand Side by the TS solver

the previous section, the `PetscFV` and `DS` object are required for the subsequent computations used by the time stepping solver.

2. The routine `DMPlexTSComputeRHSFunctionFVM()` is used to trigger the computation of the time derivative using the Riemann solver specified using the `DS` object.

3. The algorithm for computing the flux residuals for every cell starts with the geometric data obtained from the IstinlineDMPLEX object. The gradients are then computed using the solution vector and the adjacency information. The routine responsible for the reconstruction of the gradients is encapsulated by the PetscFV object. Subsequently, if the usage of a limiter is specified, they are used during the reconstruction procedure.

4. If the gradient computation is disabled, the state variables are not reconstructed at the cell interfaces. Rather, in this case, the cell center values are directly taken into consideration by the Riemann solver.

5. A loop is passed which spans across the cell interfaces or faces. Together with the normals and the state variables on either side of the cell interface, the Riemann fluxes are calculated for every face.

6. Finally another loop, now spanning across every cell is passed. The fluxes across each of the face are accumulated and the residual value is computed. Values corresponding to the cell-centered source term can be included in this routine. The time derivative vector is then finally passed to the time-stepping solver.

## 4.5. Integration of DMPlex Example

As mentioned in section 1.6, one of the tertiary objectives was to provide a smoother ramp for the creation of time-dependent numerical models using PETSc. Therefore, a new TS tutorial was integrated into the development version of PETSc. The tutorial consists of a uniform grid, a simple advection model based on FVM, and DMPlex. A number of reasons motivated this step.

After a survey of the currently available TS tutorials available in the repository, there were not any programs dedicated to demonstrating different routines of DMPlex. Furthermore, this void is more significant for models based on FVM as there were only 2 tutorials using the numerical methodology. In both of the tutorials, advanced routines and adaptive mesh refinement were instituted. As one might imagine, there is a huge leap in complexity between a uniform solver and an adaptive grid solver. Therefore, the tutorial integrated served to fill this void. Next, the discretisation methods in the available examples are not explicitly provided. This makes it harder to understand the role of each routine. The same applies to the Riemann solver interface used by the `DS` and `TS` object.

The proposition for a new tutorial was gladly accepted by Mark Adams and Matthew Knepley of the PETSc Development team who acknowledged the gap mentioned before. In order to make changes to the repository, one has to go through the Continuous Integration process used by PETSc. This allows for multiple developers (with the appropriate authorisation) to seamlessly introduce improvement and fixes without disrupting the master branch. This integration process is implemented by creating a software pipeline containing multiple checks for the various compiler warnings based on different operating systems. Once the pipeline is passed, the new tutorial has to pass through a review done by the primary developers of the software. In this case, the review was done by developers *Mark Adams*, *Matthew Knepley*, *Jed Brown* and *Satish Balay*. The tutorial can be found in the latest version of PETSc under the directory `./src/ts/tutorials/ex52.c`

# 5

# Adaptive Grid Refinement

Using an existing implementation in the PETSc DM module, it was possible to carry out adaptive grid simulations for the advection and the SWE model. This section will focus on the mathematical details behind the adaptive grid method. The models implemented earlier were only capable of handling uniform, structured grids produced using functions in DMDA and DMPlex. To shift to adaptive grid method, the initial grid produced by `DMPLEXCreateBoxMesh()` is refined or coarsened according to various adapting criteria that are discussed later in section 5.5. But first, DMPlex, the module responsible for handling unstructured grid data is described in detail in the following section.

## 5.1. DMPlex

The subsystem DMPlex of the module DM is used to manage all unstructured grids in PETSc [8]. As explained in section 4.2.5, DMPlex supports the management of data of non-conformal, unstructured grids which are to be used in the adaptive grid algorithms. The prime advantage offered by DMPlex is that it recognizes every component of the mesh composed of points related to each other. This enables the recycling of algorithms across multiple dimensions without a radical shift in mesh management routines.

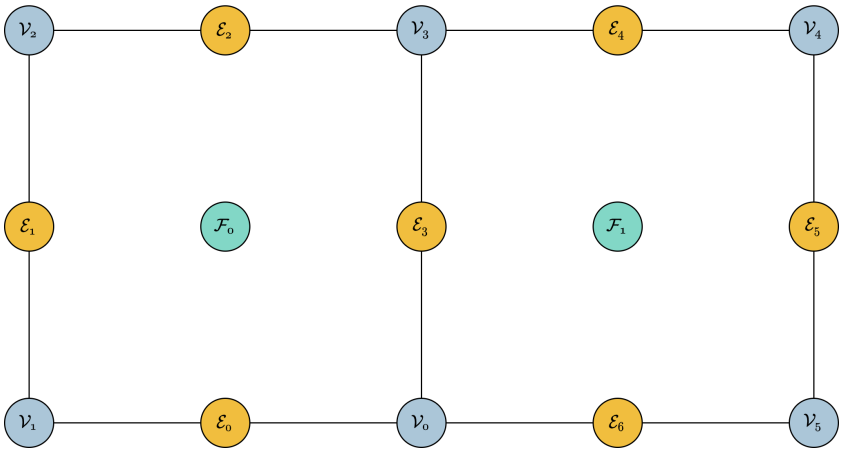



Figure 5.1: Hasse diagram of a 2D quadrilateral mesh where ($\mathcal{E}_i$) is the edge, ($\mathcal{V}_i$) is the vertex, and ($\mathcal{F}_i$) is the face for $i \in [0, 5]$

### 5.1.1. Data Layout

The points are stored as the data type `int` and their corresponding relations between each other are termed as cones. The Hasse diagram, shown in 5.1, is used to represent the relations between the points which can then be combined to form different entities in the grid. Also called a Directed Acyclic Graph, the Hasse diagram in DMPlex is composed of layers corresponding to the different entities in the grid. According to the conventions used by PETSc, the grid entities are stored consecutively and

the order varies according to the dimension. For instance, in a two-dimensional grid, the points of entities are stored in the order: faces ($\mathcal{F}$), vertices ($\mathcal{V}$), and edges ($\mathcal{E}$).

To couple the discretisation with the specified PDE, an object called `PetscSection` is used. Essentially, it provides a data layout to store and associate the degrees of freedom with each point in space. Therefore, with the `DM` object in hand, a `PetscSection` object is created with parameters specific to the PDE being solved.

## 5.2. Partitioning

The partitioning of the grid is required to take place before the actual numerical operations start. Therefore, it can be included in the pre-processing steps. It is thus vital to note that the partitioning of the grid is a prerequisite for creating parallel data structures. PETSc outsources much of this workload by using parallel partitioning algorithms provided by ParMETIS [35]. An interface developed by George Karypis allows for the use of the parallel mesh distribution algorithms in PETSc.

As per version 3.13 (June 2nd, 2020), PETSc does not feature dynamic partitioning that allows the balancing of loads by sending and receiving matrix and vector elements between the different processes. This is especially useful when one is trying to dynamically refine or coarsen the grids. It would promote the recycling of older data structures to create updated structures. In the absence of this route, adaptive mesh refinements in PETSc takes place through recreating the data structures from scratch.

## 5.3. Quad-Tree

The quad-tree is a natural extension to a uniform Cartesian grid. This grid, however, is limited to be composed of only square elements or cells. A quad-tree adaption is represented in figure 5.2.

Here, cell **a** is considered to be at the initial state of the grid with refinement level, say, 1. Consequently, by using a given adaptive criteria, the initial grid (cell **a** in figure 5.2) is adapted or refined to 4 square cells. The cells, thus have a refinement level of 2 (for instance cell **b**). In the next adaption step, the diagonal cells are further refined into 4 more cells. Cell **c** has a refinement level of 3 in the 3rd adaption cycle. In the final step, after another round of adaption, we obtain a higher refinement level. The same logic in reverse can be used for coarsening.

Note that in order to keep the model stable for the different configurations of the grid, it is necessary to maintain a 1:2 refinement and coarsening ratio. Thus, no neighbouring cells in both the diagonal and orthogonal directions can differ by more than one level of refinement. If the condition is violated, then it might be the case that the interpolation errors would dominate the errors introduced by the model. Therefore, this may ultimately lead to either an invalid solution or instability during run time.

## 5.4. Interpolation Methodology

In this section, the grid-adaption routines used by PETSc is described in detail. It must be noted that the methodology starts from the assumption that the grid partitioning and the formation of the data structures ”post-adaption” are already available. Partitioning algorithms, thus, would not be discussed here as it is beyond the scope of this project. Two different adaption scenarios are considered in this section; a grid being refined and a grid being coarsened.

One of the prime advantages of quad-trees is that it allows one to form a parent-child relationship between two different cells with different refinement levels. To illustrate this property, consider figure 5.3. Suppose we have 2 `DM` objects containing the grid data, namely $DM_{in}$ - the ”pre-adaption” grid and $DM_{out}$ - the ”post-adaption” grid. Furthermore, for both of the grids, $DM_{ref}$ is considered as the reference grid. Reference grids are typically uniform containing cells at a specified initial refinement level. It can be seen that using the parent-child hierarchy diagram, one can derive the relation between the cell $\{2\}$ (parent) and cells $\{5,6,7,8\}$ (children). This information enables efficient interpolation routines while refining or coarsening a given region of the domain. These techniques are further elaborated in the following sections.

### 5.4.1. Refining

The adaptive mesh refinement algorithm also must provide routines to compute the new state vector, say $\mathcal{U}_i^n$ on the adapted grid at timestep n of cell i. In the case of refining, a linear interpolation technique

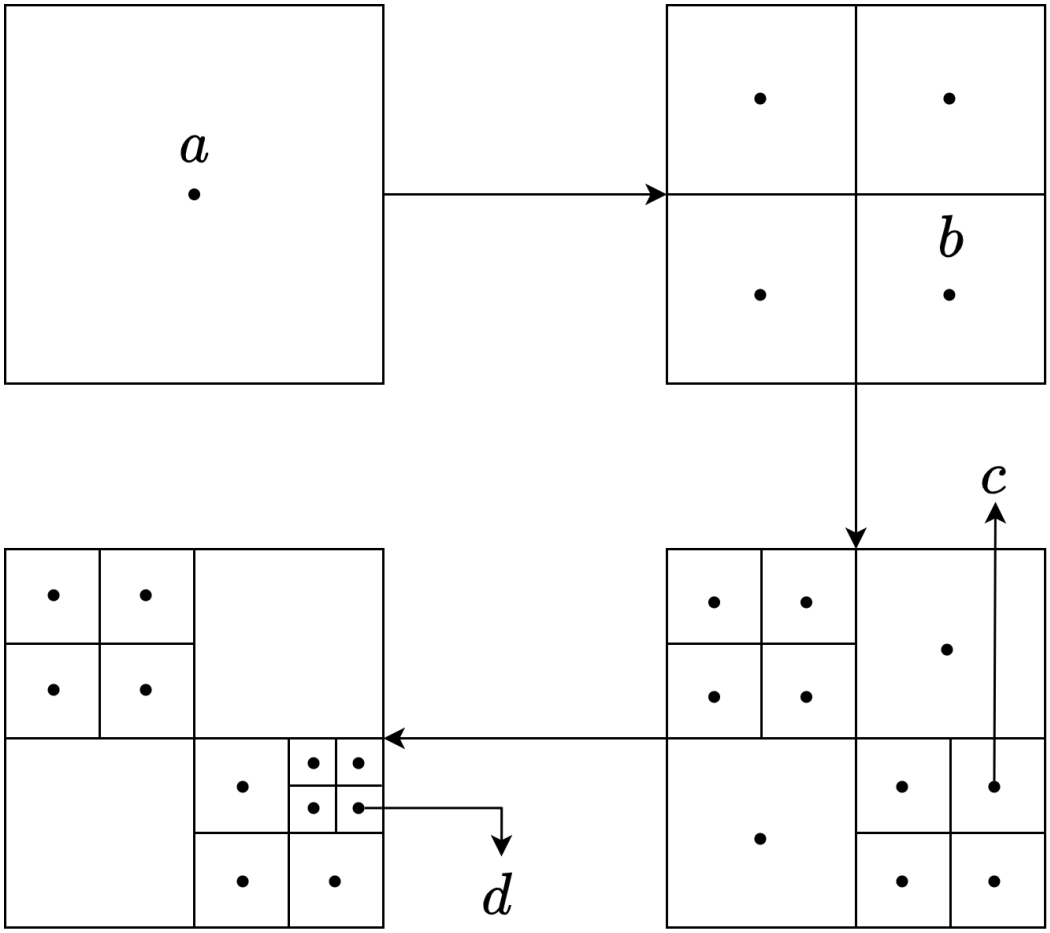



Figure 5.2: Illustration of a quad-tree structure

is employed to reconstruct the cell values for the post-adaption grid. Consider figure 5.3, the pre- and post-adaption grids are $DM_{in}$ and $DM_{out}$ respectively.

Using the `DM_in` and `PetscFV` objects, the gradients of the state vector are computed using `DMPlexComputeGradientFVM()` for the pre-adaption grid. Furthermore, centroids of the cells are known for both the pre- and post-adaption grids. Using this information, a linear interpolation, similar to the one mentioned in section 3.5 can be carried out and the state vector of the 'daughter' cells can be computed.

To further illustrate this, consider cell $\{2\}$ in $DM_{in}$. It's refinement level is increased by 1, thus resulting in the formation of 4 more 'daughter' cells $\{5, 6, 7, 8\}$. Now consider the reconstruction of the state variables at cell centre $8$. Suppose the centroid of $2$ is at $X_2 = (x_2, y_2)$ and the centroid of $3$ is at $X_8 = (x_8, y_8)$, then

$$D = (X_8 - X_2) = (\Delta x, \Delta y)^T \tag{5.1}$$

If $\mathcal{G} = (\mathcal{G}_x, \mathcal{G}_y)^T$ are the gradients of state vector $\mathcal{U}_2$ of cell $2$ in the x and y directions, then the state vector $\mathcal{U}_8$ at cell $8$ can be computed by

$$\mathcal{U}_8 = \mathcal{U}_2 + \mathcal{G} \,.\, D \tag{5.2}$$

### 5.4.2. Coarsening

Due to the hierarchical structure of the grid, coarsening or reducing a refinement level is achieved very efficiently. When the grid is coarsened, the grid associated with the 'daughter' cell is essentially rolled back to the grid with its parent cell configuration. Therefore, the state vector of the newer, coarser

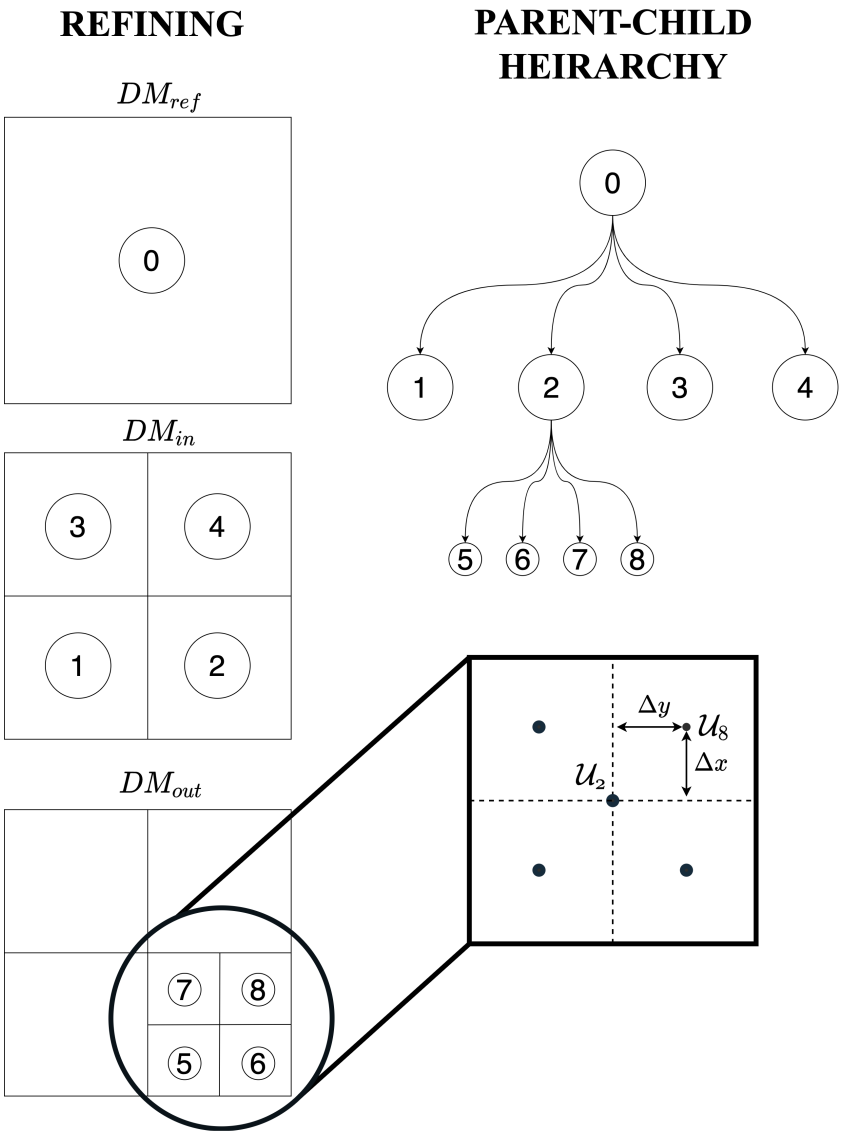



Figure 5.3: Refinement of a DM object and interpolation of state vector

cell can be obtained by directly injecting the value already owned by the parent cell. In figure 5.4, cells $\{1, 2, 3, 4\}$ are coarsened to obtain cell $0$. Here, there is no need for further reconstruction of state vector at cell $0$ as the hierarchical model of the grid allows for the direct injection of the value. However, if the final refinement level of the cell falls below that of the reference grid, then reconstruction would be necessary.

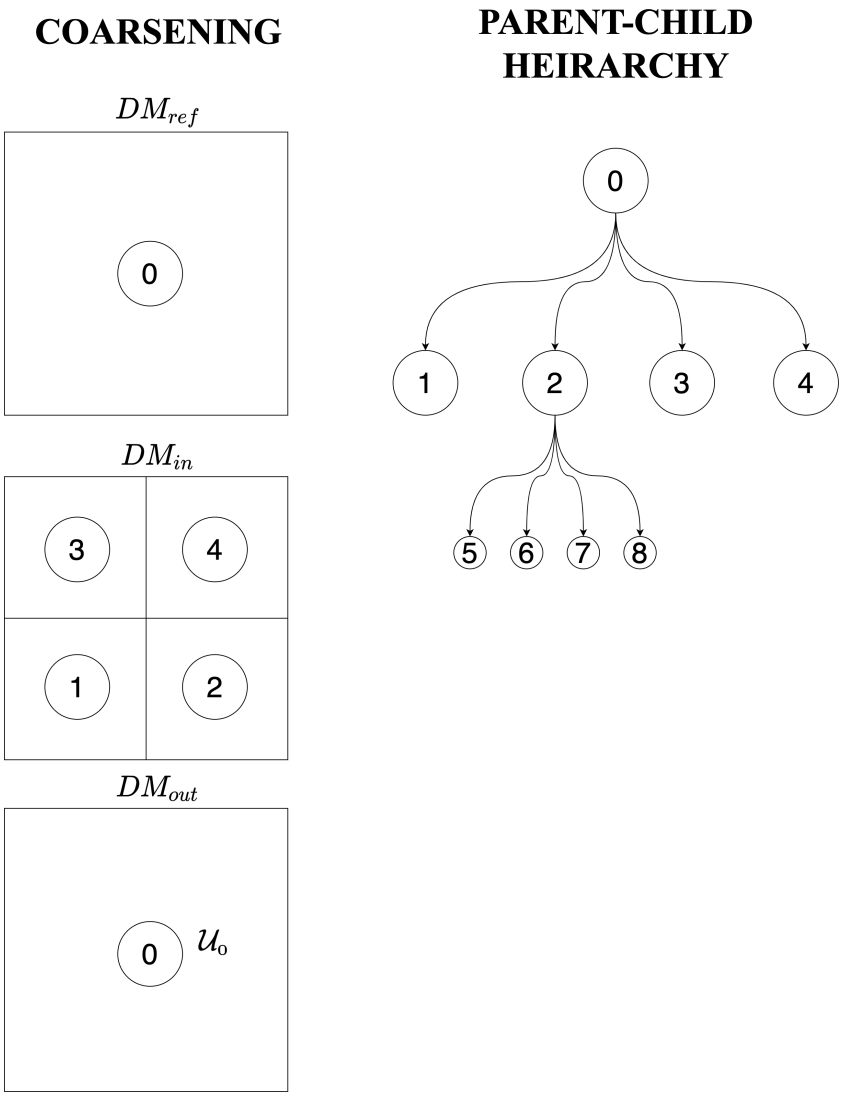



Figure 5.4: Coarsening a DM object and injection of state vector

## 5.5. Adapting Criteria

The main attraction towards pursuing an adaptive grid approach is that it allows the increase in the resolution of the solution in specific areas of interest. The areas of interest can be dictated by different parameters and it changes over time. The quantifying parameter will henceforth be called as the adapting criterion $\Theta$. The type and magnitude of $\Theta$ are critical to the accuracy of the final solution. In addition, the choice is also critical to the computational resources required.

When considering the threshold values of the adapting criteria ($\Theta_1$ and $\Theta_2$), we consider either an absolute value or a relative value. They are described more in detail below.

- Absolute Value: In this case, fixed values of $\Theta_1$ and $\Theta_2$ are considered when deciding whether to adapt a given cell. The decision is carried out using the following criteria:

  1. Refine the cell when $\Theta > \Theta_1$.

2. Coarsen the cell when $\Theta < \Theta_2$.
3. Keep the same refinement level otherwise.

Here, if the criterion falls above $\Theta_1$, the cell is refined. On the other hand, if it falls below $\Theta_2$, the cell is coarsened. The threshold values are dependent on the scenario and are thus found using a trial and error process. It varies according to a given scenario and domain.

- Relative Value: In this project, a relative value of $\Theta_{max}$ is used instead. The decision of adaption is now based on the ratio of the $\Theta$ of a cell with respect to the highest value of $\Theta$ at that particular time step of all the cells in the domain. Thus, refining or coarsening is carried out using the following logic:

  1. Refine the cell when $\frac{\Theta_{i,j}}{\Theta_{max}} > \Theta_{r1}$.
  2. Coarsen the cell when $\frac{\Theta_{i,j}}{\Theta_{max}} < \Theta_{r2}$.
  3. Keep the same refinement level otherwise.

  Here, if the criterion falls above $\Theta_{r1}$, the cell is refined. On the other hand, if it falls below $\Theta_{r2}$, the cell is coarsened. Using this strategy reduces the level of arbitrariness when choosing the threshold values of the criteria. Although this is a better approach to choose when drawing comparisons between various types of criteria, it introduces certain challenges. One of the challenges being that there might be unwanted adaption for minor perturbations in some parts of the domain.

The choice for the formulation of adapting criteria depends on the physical system under consideration. Therefore, the formulation for both the scalar advection and the shallow water equations are presented in separate sub-sections.

### 5.5.1. Criteria for Scalar Advection Equation

Due to the scalar nature of the equation, $\Theta$ is easily identified. The gradients of the state variable in both directions are considered. The formulation is thus given below:

$$\Theta = \sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2}$$

Criteria for Shallow Water Equations

In this case, a number of potential parameters are identified for the adaption of the grid. The gradient of water height is a good indicator of change in the solution. Nevertheless, changes in velocity could also indicate the wave propagation. This might lead to a solution that could better capture the wave-front. In view of these different factors, three different formulations are considered and presented below.

1. Gradients of Height : Here, the only deciding factor for the adaption is based on the change in heights of the solution.

$$\Theta = \sqrt{\left(\frac{\partial h}{\partial x}\right)^2 + \left(\frac{\partial h}{\partial y}\right)^2}$$

2. Average of the gradients of height and tensile components of velocity: The change in velocities are taken into account in the criterion. Nevertheless, the sheer gradients of the velocity are neglected.

$$\Theta = \frac{1}{2}\left(\sqrt{\left(\frac{\partial h}{\partial x}\right)^2 + \left(\frac{\partial h}{\partial y}\right)^2} + \sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2}\right)$$

3. Sum of the gradients of the state vector: The sum of the squares of the gradients of the state vector is considered in this formulation. Note the inclusion of the sheer gradients of the velocity.

$$\Theta = h_x^2 + h_y^2 + u_x^2 + u_y^2 + v_x^2 + v_y^2$$

After a number of tests, it was found that criteria 2 and 3 are highly sensitive, especially at the boundaries of the domain. This could be attributed to the contribution of the sheer gradients of the velocity. Tests with criteria 1, on the other hand, do not show this behaviour at the boundaries. Furthermore, it was found to accurately identify the wave-fronts and other areas of interests in the domain. Therefore, the gradients of height were chosen as the criteria of choice.

## 5.6. Implementation in PETSc

As described in section 4.2.5, the module DMForest contains a suite of objects and subroutines that facilitates the handling and usage of hierarchical meshes. Nevertheless, additional steps are to be made to pre-process the `DM` object to treat the special artifacts of non-conformal meshes such as dangling edges. Furthermore, the procedure involved in the adaptive mesh refinement routine, `adapttotolerance()` is described in the following sections.

### 5.6.1. Algorithm for the Adaption Routine

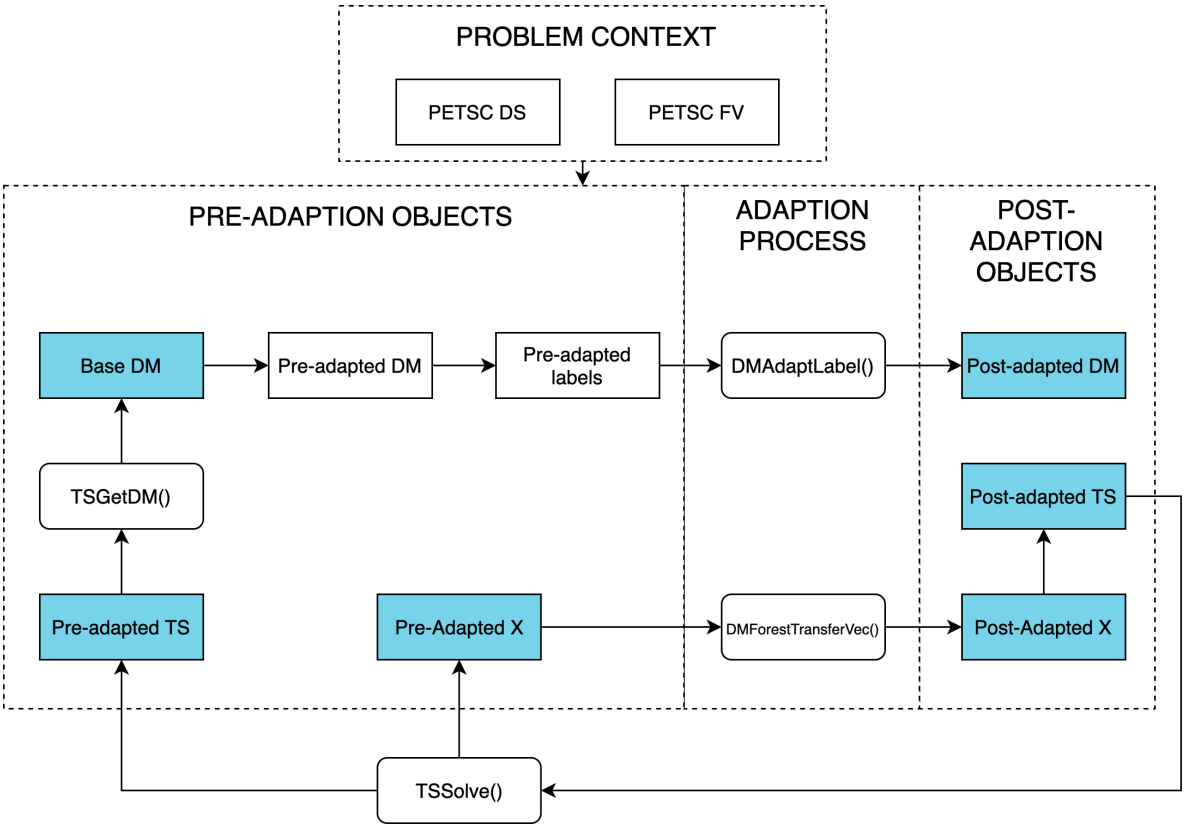



Figure 5.5: Flow of operations for adaption

The algorithm of the adaptive mesh refinement method is shown in figure 5.5. The subroutine `adapttotolerance()` is responsible for carrying out the adaption process according to a selected criterion. Therefore, all of the routines explained here are performed by calling this function.

The process starts with the pre-adapted TS object. At the start of the process, the pre-adapted solution vector `X` and pre-adapted `TS` object are available. The `DM` contained by the pre-adapted `TS` object can be considered as the base `DM` that needs to adapted according to a given strategy. A pre-adapted `DM` object is created by duplicating the base `DM` as a measure to not change or disrupt the pointers of the existing `DM`. According to the adaption criteria $\Theta$ (section 5.5), the cells in the grid are marked with the following labels:

- `DM_ADAPT_LABEL_REFINE` : Label given to increase the refinement level by 1.
- `DM_ADAPT_LABEL_COARSEN` : Label given to decrease the refinement level by 1.

- `DM_ADAPT_LABEL_KEEP` : Label given to maintain the same refinement level.

Once the labels are assigned to the pre-adapted `DM`, it can be adapted using the function

- `DMForestSetMaximumRefinementLevel()` : The maximum refinement level that any cell in the grid can attain.
- `DMForestSetMinimumRefinementLevel()` : The minimum refinement level that any cell in the grid can attain.
- `DMForestSetInitialRefinementLevel()` : The initial refinement level of the reference grid. This is provided during the creation of the reference grid.

Once the post-adapted `DM` is computed, the solution vector `X` (pre-adapted vector) can also be transferred to the new `DM`. This process is responsible for the re-interpolation of the solution vector to the post-adapted `DM`. Using the updated solution vector, an updated TS object can also be obtained. This is ultimately used to march the solution to the next time-step.

### 5.6.2. Process Description

With the help of the description provided by the previous sections, the flow of process associated with the adaptive grid method in the solver is explained in this section.

1. The process starts with a uniform grid with a given initial refinement level. The solution vector is then created with the prescribed initial condition.
2. An initial adaption procedure is carried out in order to provide an adequate resolution to the areas of the grids that requires it. For instance, in the case of a dam break problem, it is essential to have highly refined cells in the region which contains the hydraulic jump. The initial adaption can be carried out using `adapttotolerance()`. However in this step, the routine `DMForestTransferVec()` (see figure 5.5) is not employed to avoid interpolation errors. Instead, the exact initial solution is projected onto the updated solution vector. The initial adaption procedure is usually limited by a specified number of iterations in light of the computational resources and memory used.
3. After the initial adaption is completed, the solution is marched forward by a time-step using `TSSolve()`. Subsequently, the time-stepping procedure is carried out by the algorithm provided in section 5.6.1. The user can also set the number of adaption iterations for each time-step. The recommended value is 1 adaption iteration per time-step *after* the initial adaption step.
4. Once the maximum time is reached, the solution vector X is thus post-processed and outputted in a specified format.

# 6

# Numerical Results

## 6.1. Scalar Advection Equation

The purpose of this test to evaluate the performance of an advection solver built using DMDA (Finite Difference Method) and DMPLEX (Finite Volume Method) and to verify the numerical order of accuracy. The description of the test cases and the respective results are given in the upcoming sections. For all of the test cases, the absolute error $\epsilon$ is chosen as the barometer. Suppose $i \in [1, n]$, then the L1 error norm is computed using

$$\|\hat{\epsilon}\|_1 = \sum_{i=1}^{n} \epsilon_i A_i \tag{6.1}$$

where $A_i$ is the area of the cell and $\hat{\epsilon}_i = u_{num} - u_{exact}$ is the relative error. Here $u_{num}$ and $u_{exact}$ are the numerical and exact solutions respectively.

In order to measure the efficacy and accuracy of the model, a simple initial condition was prescribed. The exact solution to the problem is known, and the error is thus computed using numerical and exact solutions. For the initial condition, a smooth bump is projected on the domain $u \in \Omega : \Omega \subset [0,1] \times [0,1]$.

$$u_0(x,y) = \begin{cases} 1 + e^{-30r}, & \text{if } r < 0.025 \\ 1, & \text{otherwise} \end{cases}$$

where $r = (x-0.5)^2 + (y-0.5)^2$. For the above initial condition, the exact solution is given as:

$$u(x,y,t) = \begin{cases} 1 + e^{-30r(t)}, & \text{if } r(t) < 0.025 \\ 1, & \text{otherwise} \end{cases}$$
$$r(t) = (x - (0.5 + at))^2 + (y - (0.5 + bt))^2$$

Here a and b are considered as known constants. Parametric studies were conducted separately using the DMDA and DMPlex solvers.

For this scenario, a transmissive boundary condition was imposed on all the 4 boundaries. Consider a one dimensional grid of size n, where cells $u_0$ and $u_{n+1}$ are considered as ghost points, the transmissive boundary condition can be defined as:

$$u_0 = u_1$$
$$h_{n+1} = h_n,$$

In this condition, the flow is meant to freely exit the domain without any disruption. Hence they can also be called open boundaries.

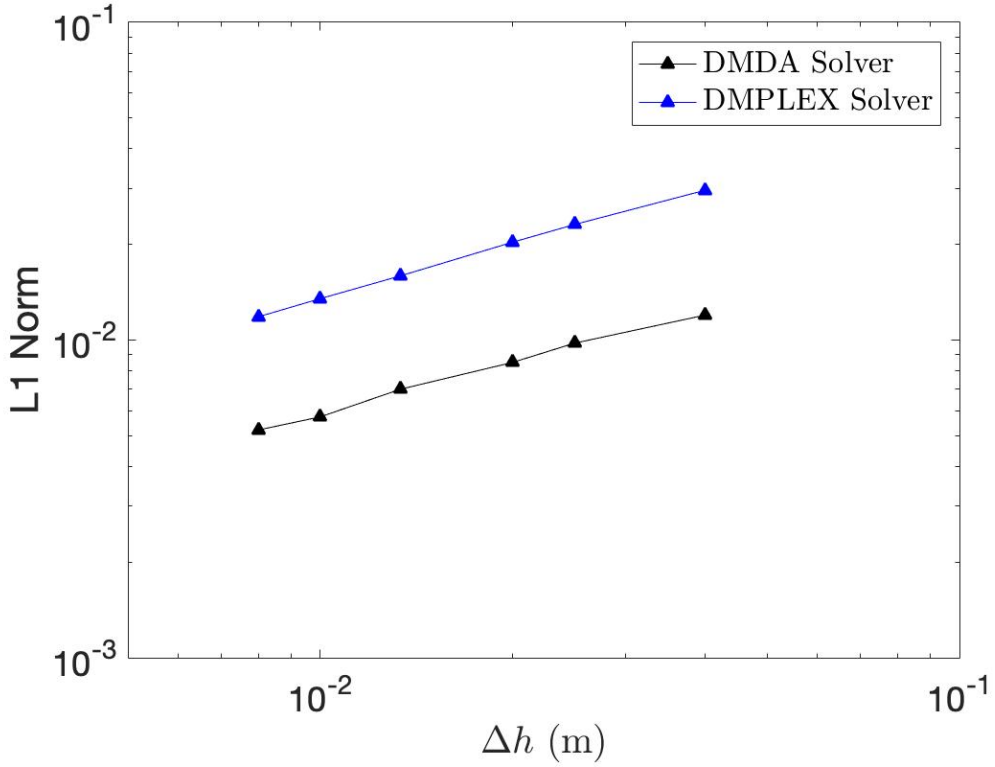



Figure 6.1: Varying the meshwidth by keeping a constant timestep

### 6.1.1. Error Comparison

As mentioned previously, the simple advection model was built using both the FDM and FVM approaches. As explained in section 4.2.5, DMDA contains routines to handle structured Cartesian grids. On the other hand, DMPlex can handle unstructured and/or nor-conformal grids. To make a direct comparison, only structured, uniform grids are taken into consideration. As mentioned in section 3.1, the Forward Euler method is used to march in time and a first-order upwind method is used for the spatial discretisation. Therefore, both of the methodologies are first order in space and time and are equivalent in mathematical terms.

| **Mesh-width (m)** | **L1 Norm** | **Order** |
|---|---|---|
| 1/25 | 0.029618 | 0.93 |
| 1/50 | 0.020330 | 0.95 |
| 1/75 | 0.015916 | 0.78 |
| 1/100 | 0.013502 | 0.86 |
| 1/125 | 0.011837 | 0.89 |

Table 6.1: Error norms of the *DMPLEX* solver

| **Mesh-width (m)** | **L1 Norm** | **Order** |
|---|---|---|
| 1/25 | 0.011983 | 0.70 |
| 1/50 | 0.008507 | 0.91 |
| 1/75 | 0.006998 | 0.81 |
| 1/100 | 0.005731 | 0.83 |
| 1/125 | 0.005211 | 0.86 |

Table 6.2: Error norms of the *DMDA* solver

Assuming the temporal discretisation error is negligible, a timestep of $0.005\,sec$ is taken. The mesh-widths are however varied from $1/25\,m$ to $1/125\,m$. The L1 norms are computed using (6.1) for each of the cases. Figure 6.1 shows the trends in errors for both of the solvers. The difference in the errors shown between the solvers can be attributed to the variation in the routines used by the `TS` and/or the `DM` modules. Nevertheless, the numerical order of accuracy in space is close to the theoretical order of accuracy as shown in tables 6.1 and 6.2.

### 6.1.2. Scaling Tests

In order to evaluate the parallel efficiency of the user-defined and pre-built algorithms, a strong scaling test is performed. A strong scaling test is conducted by constraining the grid to be of the same mesh-width while varying the number of workers. All of the tests were conducted on an Intel Core™ i7-7700HQ CPU $(4 \times 2.80\,Ghz)$ with $31.2GB$ of on-board RAM. The problem description remains the same as the ones mentioned in the previous section. A mesh-width is chosen such that a perfect load balancing can be achieved across the processor range. Therefore, a $240 \times 240$ uniform Cartesian grid is used where the final time is $T = 50\,sec$ and the time-step is $dt = 0.001\,sec$.

Furthermore, the parallel efficiencies are computed using

$$\text{Speedup Ratio} = \frac{T_1}{T_h} \tag{6.2}$$

$$\text{Parallel Efficiency} = \frac{T_h}{h\,.\,T_1} \tag{6.3}$$

where $T_h$ is the time taken by using $h$ processors and $T_1$ is the time taken by a single processor.

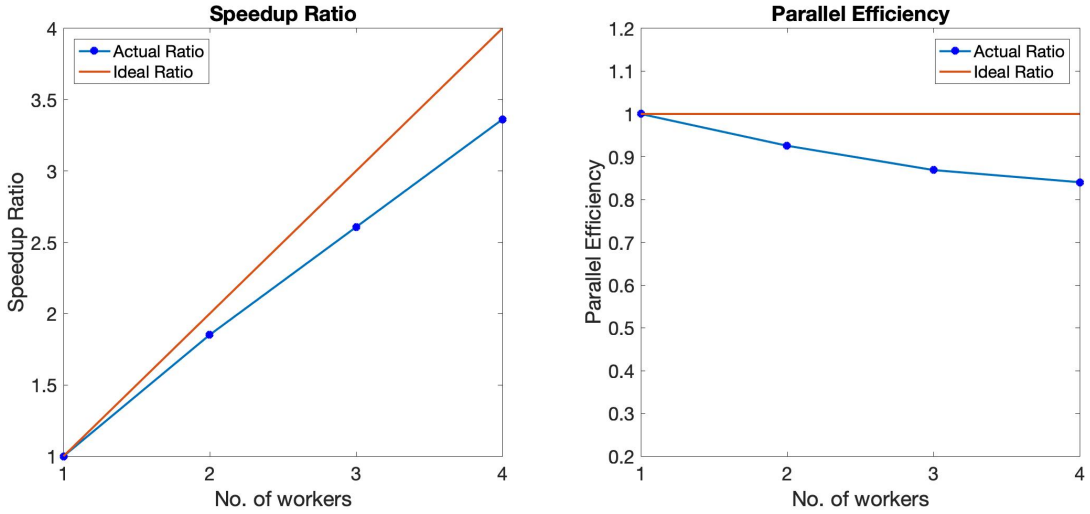



Figure 6.2: Scalability of the DMDA solver

Using the inbuilt profiling routines, the number of FLOP(s) (Floating point operations) for the two solvers were evaluated. Table 6.3 and 6.4 provides the FLOP count, memory consumption and time consumption percentages *of some of the major routines* when using 4 processors. Please note that the percentages are meant to represent the weights of each routine. Additionally, as some of the routines are subsets of other routines, considerable overlap can be found in the memory and FLOP percentage values.

As mentioned in 4.2.5, the DMDA is highly optimised for cell-center based finite difference methods. This can be noticed in the efficiencies shown in figure 6.2. Nevertheless, to implement the Riemann solver and use non-uniform meshes, it was imperative to make a shift to DMPlex. However, for the current numerical methodology employed, the usage of DMPlex can be considered inefficient. This is demonstrated in figure 6.3. A large portion of the computation time is consumed by routines that compute residuals over each cell (table 6.1). By looping over the faces in the domain, there is a significant increase in the communication overhead when compared to the DMDA solver. Furthermore, the majority of the FLOP(s) is consumed by time-stepping routine

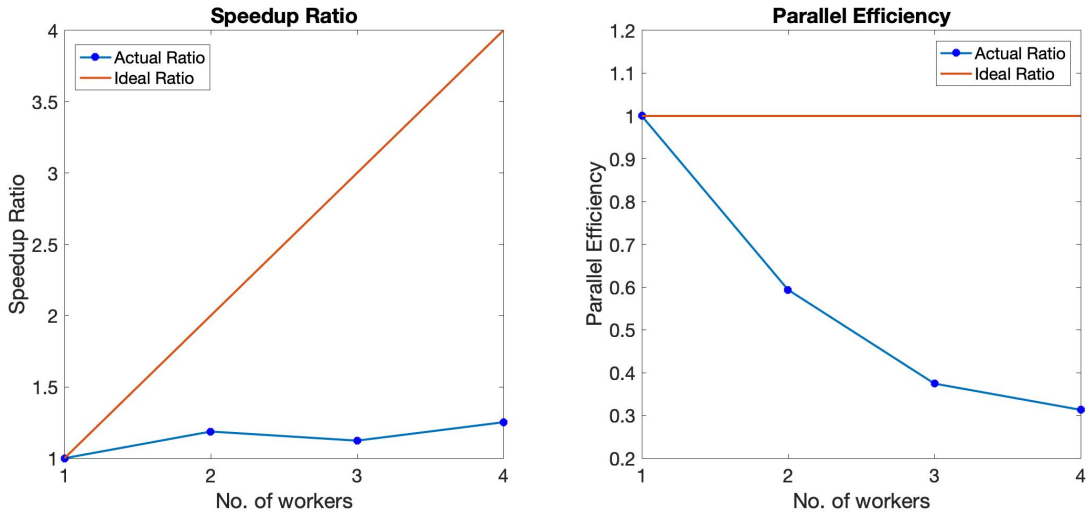



Figure 6.3: Scalability of the DMPlex solver

| Routine | % Time | % FLOP | % Memory |
|---|---|---|---|
| Norm Computation | 23 | 13 | 0 |
| Vector Operations | 12 | 13 | 0 |
| Vector Partitioning | 7 | 0 | 100 |
| Time Stepping | 74 | 87 | 99 |
| RHS Evaluation | 58 | 73 | 99 |

Table 6.3: Profiling of the DMDA Solver

| Routine | % Time | % FLOP | % Memory |
|---|---|---|---|
| Residual Evaluation | 76 | 0 | 33 |
| Grid Operations | 33 | 13 | 100 |
| Norm Computation | 4 | 43 | 0 |
| Vector Operations | 1 | 43 | 0 |
| Time Stepping | 95 | 56 | 100 |
| RHS Evaluation | 93 | 13 | 100 |

Table 6.4: Profiling of the DMPlex Solver

## 6.2. Shallow Water Equations

A number of benchmark cases are available to validate and benchmark the shallow water model. The tests were conducted to evaluate the solver for the validity of its solutions across the domain. Typically this information serves a key role in the risk analysis report for the civil departments to produce flood inundation maps.

One of the primary tests includes the dam break problem. A dam break scenario can be regarded as an idealized condition that is extremely rare to occur in a real-life setting. However, surpassing all the odds, a total failure occurred in 1959, when the Malpasset dam ruptured, subsequently ravaging human and animal life. This event, being rare yet disastrous, poses a hard challenge for numerical models. Recognizing the significance of this scenario, a CADAM European workshop was created to experimentally simulate the flow features of the dam break problem. Thus, using this data, the results from the model are validated in the following section.

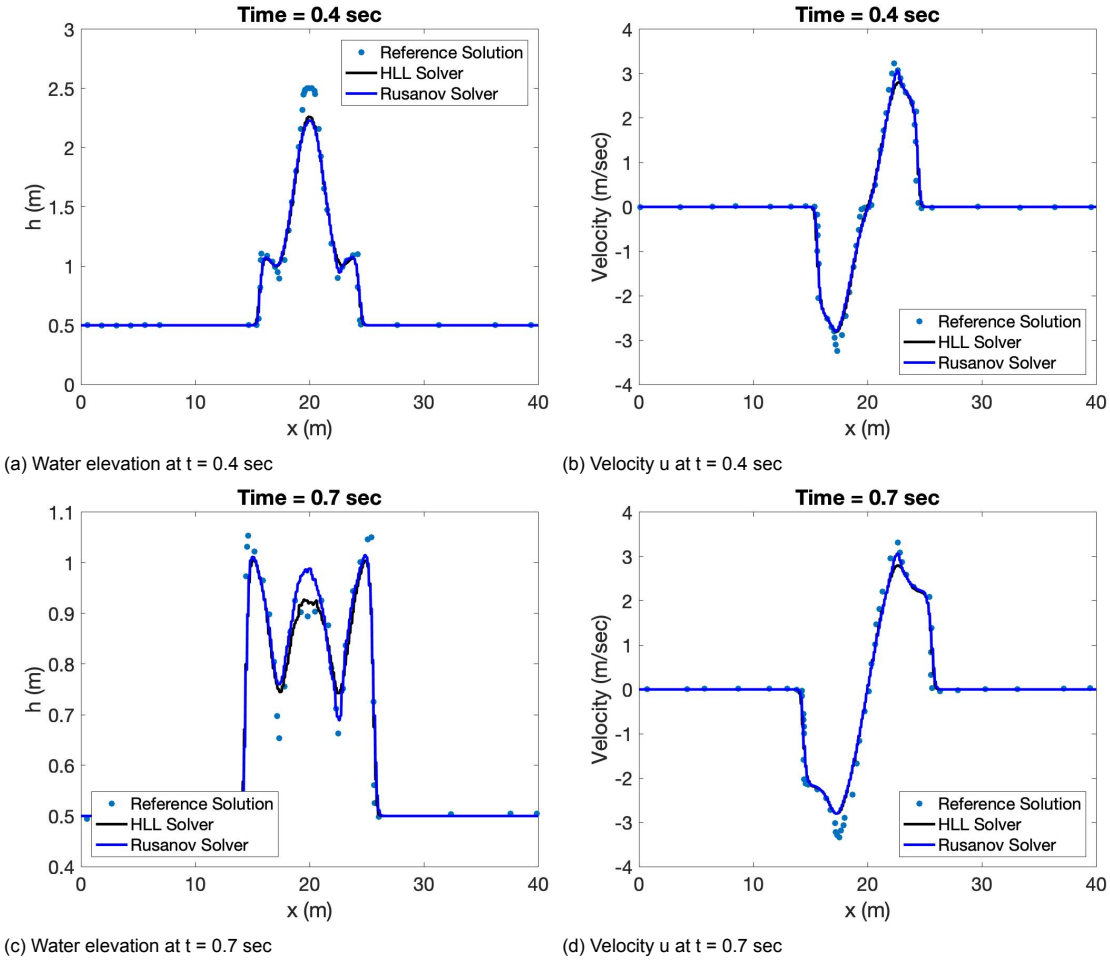
### 6.2.1. Circular Dam Break

This is an idealized scenario [54] with a completely flat topography. It is an illustrative case to study the wave propagation during the collapse of a circular wall of water. A square domain of dimensions $40\,m \times 40\,m$ is completely wet with a specified water height. In this scenario, a circular patch at the

center of the domain $(x^* = 20\,m\,,\, y^* = 20\,m)$ contains water at an elevated height. Due to no source terms, the friction and bed slope terms are neglected. Furthermore, the entire domain is considered to be at rest at $t = 0$. The initial condition is given by:

$$h(x, y, t = 0) = \begin{cases} h_{dam} = 2.5, & \text{if } (x - x^*)^2 + (y - y^*)^2 \leq r^2 \\ h = 0.5, & \text{if } (x - x^*)^2 + (y - y^*)^2 \geq r^2 \end{cases}$$

Here, $r = 2.5\,m$ is the radius of the cylindrical section of water. Assuming an infinitesimal film covering the cylindrical section of water at first. The dam break is thus initiated once this film is removed instantaneously such that the entire section of water starts gaining momentum due to gravity.




(a) Water elevation at t = 0.4 sec

(b) Velocity u at t = 0.4 sec

(c) Water elevation at t = 0.7 sec

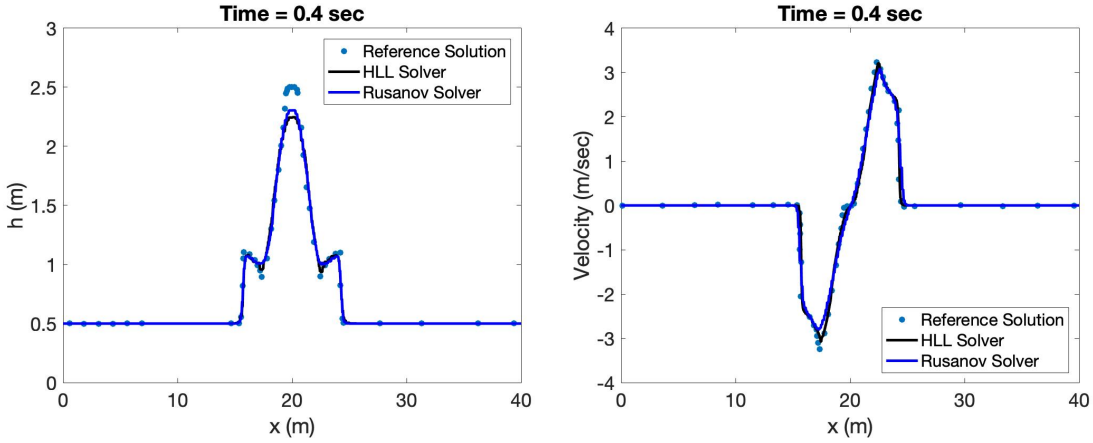
(d) Velocity u at t = 0.7 sec

Figure 6.4: Circular Dam break test on a uniform grid

Two tests were conducted and the heights and velocities of the water column at the center cross-section of the domain were captured at a given set of times. Furthermore, the tests were conducted by using the Rusanov and HLL approximate Riemann solvers explained in sections 3.4.1 and 3.4.2 respectively. For both the tests, the 2nd order accurate Runge-Kutta method was chosen as the time integrator and the CFL value was set to 1. Specific parameters of the two tests are described more in detail below.

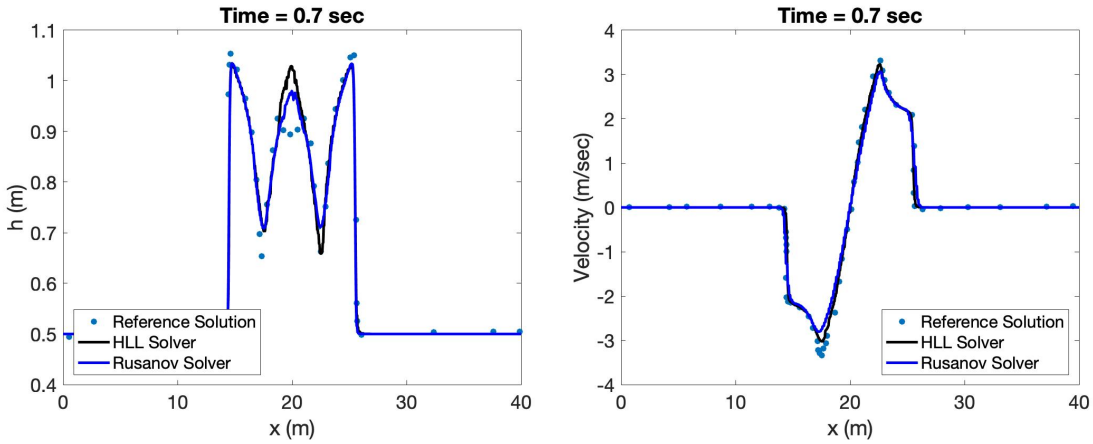
1. *Test on a uniform grid*: For this test, a fine uniform grid of mesh-width $\Delta x = 1/300m$ and $\Delta y = 1/300m$. The minmod limiter was employed for this test. The results of this test are shown in figure 6.4.

2. *Test on a time adaptive grid*: In this test, the AMR solver was employed. The mesh width of the grid before adaption is set to $\Delta x = 1/24m$ and $\Delta y = 1/24m$. At the initial time $t = 0$, the uniform




(a) Water elevation at t = 0.4 sec

(b) Velocity u at t = 0.4 sec




(c) Water elevation at t = 0.7 sec

(d) Velocity u at t = 0.7 sec

Figure 6.5: Circular Dam break test on time adaptive grid

grid is further adapted in 25 iterations based on the gradients of the initial condition. Furthermore, the maximum refinement level that a cell can attain during the simulation was set to level 7. The grid adaptivity is set up such that the cell is refined when the relative threshold $\Theta_{rel} > 0.6$. On the other hand, the cell is coarsened when $\Theta_{rel} < 0.1$ or kept at the same refinement level when it falls in this interval. The results of this test are shown in figure 6.4.

Furthermore, in this scenario, 2 boundary conditions are implemented. Consider a one-dimensional grid of size n where cells with indices $0$ and $n+1$ are considered as ghost points, the boundary conditions can then be defined as:

1. *Transmissive Boundary*: In this condition, the flow is meant to freely exit the domain without any disruption. Hence they can also be called open boundaries. The state vector can be computed using:

$$h_0 = h_1, u_0 = u_1, v_0 = v_1$$
$$h_{n+1} = h_n, u_{n+1} = u_n, v_{n+1} = v_n$$

2. *Reflective Boundary*: In this condition, the flow momentum is reversed forcing it to reflect back from the wall. The state vector can be computed using:

$$h_0 = h_1, u_0 = -u_1, v_0 = v_1$$
$$h_{n+1} = h_n, u_{n+1} = -u_n, v_{n+1} = v_n$$

At the very start of the cylindrical dam break, an initial shock is seen to travel outward from the center of the cylinder. Moreover, at $t = 0.4\,sec$, a rarefaction wave is also seen to travel in the opposite

direction towards the center. At $t = 0.7\,sec$, the rarefaction wave reflects off of the center and begins to travel outwards. This trend continues and leads to the water level dropping to the bottom at $t = 3.5/, sec$. At the same time, a secondary shock moving inwards is formed. At $t = 4.7/, sec$, the primary shock appears to reach the boundaries of the domain and the secondary shock reaches the center of the domain.

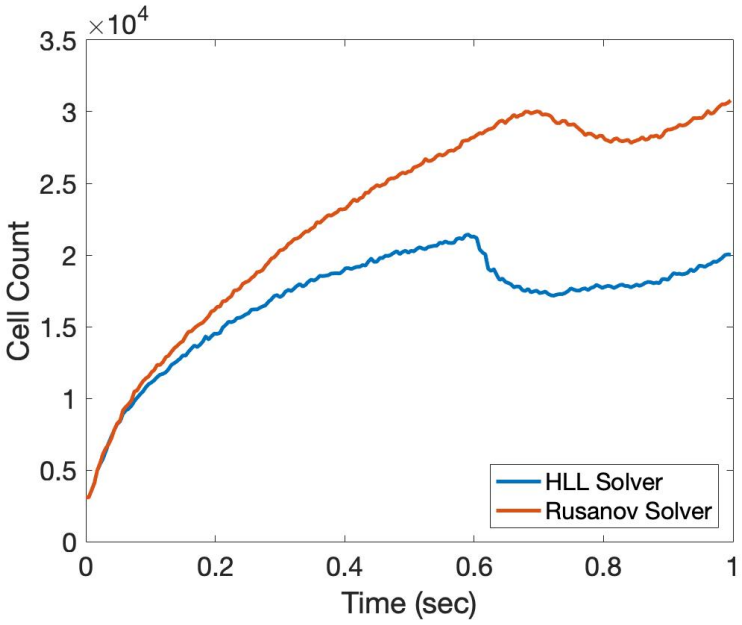



Figure 6.6: Cell count for the HLL and Rusanov approximate Riemann solvers

From figures 6.4 and 6.5, it is clear that the AMR solver achieves comparable accuracy while at the same time having average grid sizes much smaller than its counterpart. Figure 6.6 shows the evolution of the cells over time for both of the Riemann solvers. On average, the HLL solver contained 16855 cells, and the Rusanov solver solved for 22937 cells over the course of the test. Thus, it can be said, similar accuracy can be achieved with the adaptive grid solver but with far higher efficiencies. Furthermore, it can noted that although there is not a marked difference between the HLL and the Rusanov Riemann solvers in terms of accuracy, the problem of encountering negative heights might be more pronounced in the Rusanov Riemann solver. Whereas the heights computed by the HLL Riemann solver enforce positive heights with the help of a correcting factor.

### 6.2.2. Scaling Tests

Using the same hardware as mentioned in section 6.1.2, a strong scaling test is performed for the shallow water equation model for the circular dam break scenario. It is, however performed for the fixed and adaptive grid scenarios. For the purpose of obtaining an informative comparison, the grid sizes of the uniform and adaptive grid solvers are kept around the same value. However, due to the sensitivity of the adaptive grid solver, there is a difference between the mesh-width of the uniform grid solver and the average mesh-width of the adaptive grid solver. In the case of the uniform grid solver, the mesh is composed of $90 \times 90$ cells. On the other hand, in the case of the adaptive grid refinement, an average cell count of 8248 cells was found across the time of the simulation. The parallel efficiencies are again computed using the same metrics as given in equations (6.2) and (6.3). They are given by,

$$\text{Speedup Ratio} = \frac{T_1}{T_h}$$

$$\text{Parallel Efficiency} = \frac{T_h}{h\,.\,T_1}$$

For both the scenarios, the final time of $T = 1\,sec$ was used and the 2nd order Runge Kutta method was chosen as the time integration method. A fixed grid of size $120 \times 120$ was chosen for this test. For

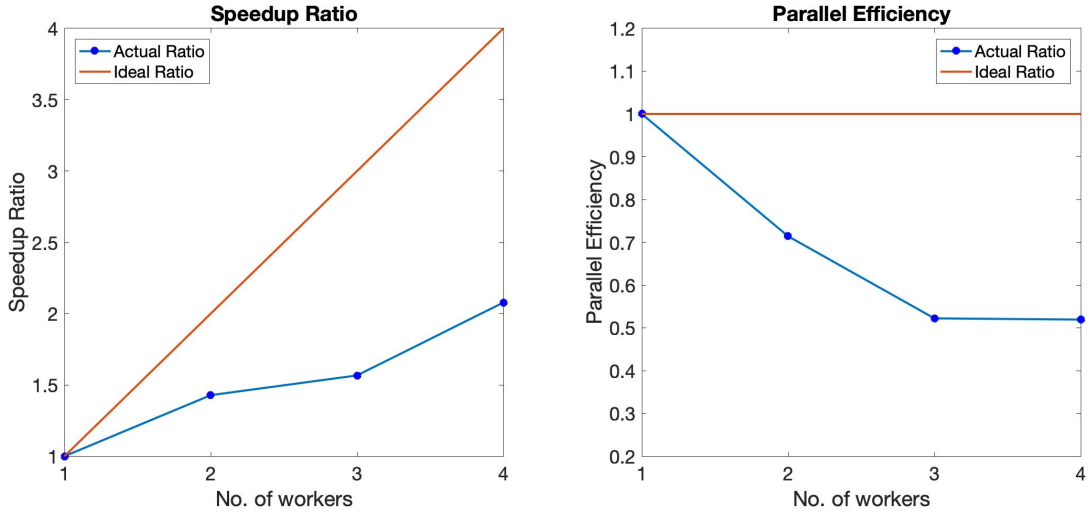



Figure 6.7: Weak scaling test of the fixed grid solver

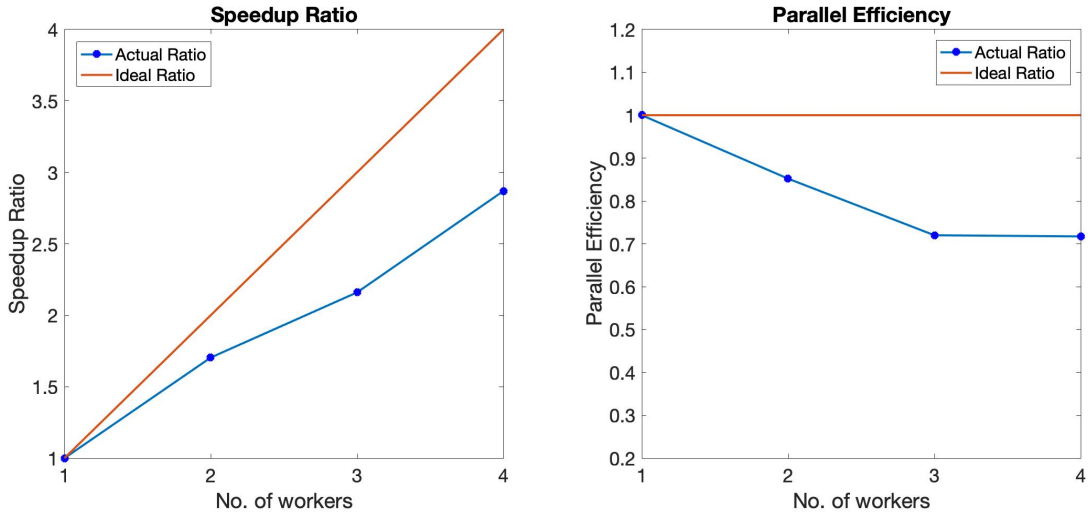



Figure 6.8: Strong scaling test of the adaptive grid solver

the adaptive grid solver, a $24 \times 24$ was chosen as the initial grid which was then passed through 25 adaption iterations. The values of the relative thresholds remained the same as in section 6.2.1. Figures 6.7 and 6.8 illustrate the scalability of the fixed and adaptive grid solvers respectively. Furthermore, the profiling results for the fixed and adaptive grid cases are provided in tables 6.5 and 6.6 respectively when using 4 processors.

| **Routine** | **% Time** | **% FLOP** | **% Memory** |
|---|---|---|---|
| Residual Evaluation | 59 | 0 | 19 |
| Grid Partitioning | 3 | 0 | 9 |
| Mesh Generation | 0 | 0 | 19 |
| Vector Operations | 1 | 58 | 0 |
| Time Stepping | 72 | 96 | 57 |
| RHS Evaluation | 71 | 2 | 57 |

Table 6.5: Profiling of the uniform grid solver

The primary difference between the fixed grid and adaptive grid solvers lies in the memory used.

| Routine | % Time | % FLOP | % Memory |
|---|---|---|---|
| Residual Evaluation | 27 | 0 | 9 |
| Intial Mesh Generation | 3 | 0 | 26 |
| Grid Adaption | 1 | 0 | 49 |
| Vector Operations | 0 | 44 | 0 |
| Time Stepping | 30 | 45 | 19 |
| RHS Evaluation | 30 | 1 | 19 |

Table 6.6: Profiling of the adaptive grid solver

Due to the tree structure of grids produced by DMForest, a large proportion of the memory is used by the adaption procedures. This is shown in table 6.6 where the grid generation and adaption take over $75\%$ of the total memory used. Furthermore, due to the decrease in the overall size of the grid, there is a significant decrease in the time spent in evaluating the residuals of each cell. For the fixed grid solver, this routine accounts for $59\%$ of the total computation time as opposed to $27\%$ for the adaptive grid solver. Due to the shift in the computational overhead from time stepping and residual evaluation routines to grid adaption and partitioning routines, a significant increase in the speedup ratio is noticed (figure 6.8).

### 6.2.3. Flow Over Bump

This test helps in the validation of the treatment of the source terms. It was first proposed by Goutal *et. al* [27] to validate a shallow water solver with varying topography. Furthermore, numerous other studies [46, 53, 59, 66] corroborated the original result with the ones provided by their studies.

The scenario involves a one-dimensional channel flow over a given parabolic hump in the domain. The length of the domain is 25 m and friction is neglected for these cases. The hump is defined by

$$z(x) = \begin{cases} 0.2 - 0.05(x-10)^2, & \text{if } 8 < x \leq 12 \\ 0, & \text{if otherwise} \end{cases}$$

Depending on the boundary conditions and the initial conditions, the steady-state solution could either be subcritical or transcritical. A one-dimensional grid with mesh-width $1/1000\,m$ was chosen for the test. A uniform grid was chosen for the tests as DMForest does not accommodate an one dimensional grid. Furthermore, the results provided by the uniform grid were deemed satisfactory. The CFL number was set to 1 and a minmod limiter was used.

First, for the subcritical solution, the initial water surface height $(h+z)$ is set to 2 m, and the discharge is set to $uh = 4.42\,m^2/sec$. The boundary conditions are given by

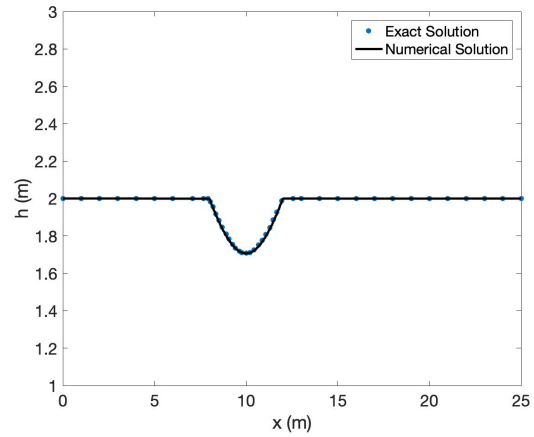
$$\begin{aligned} \text{Upstream } (x=0): &\quad uh = 4.42\,m^2/sec \\ \text{Downstream } (x=25): &\quad h = 2\,m \end{aligned}$$

The steady state solution was obtained around time $T = 40\,sec$. Fig. 6.9a and 6.9b show a good agreement between the exact solution and the numerical solution computed by the model.

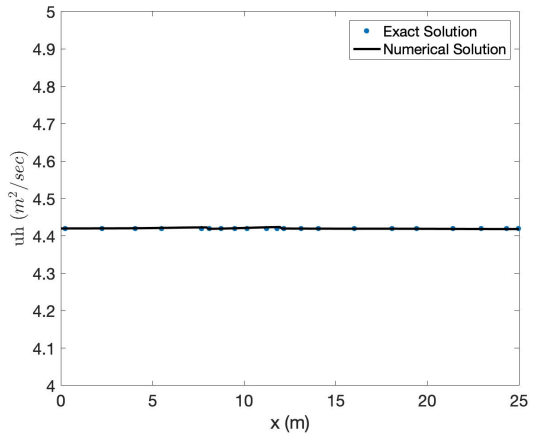
Next, for the transcritical steady-state solution, the initial water surface height $(h+z)$ is set to 0.33 m and the discharge is set to $uh = 0.18\,m^2/sec$. The test is again carried out in the same domain and on a grid with the same mesh-width as mentioned previously. The boundary conditions for this case are given by

$$\begin{aligned} \text{Upstream } (x=0): &\quad uh = 0.18\,m^2/sec \\ \text{Downstream } (x=25): &\quad h = 0.33\,m \end{aligned}$$

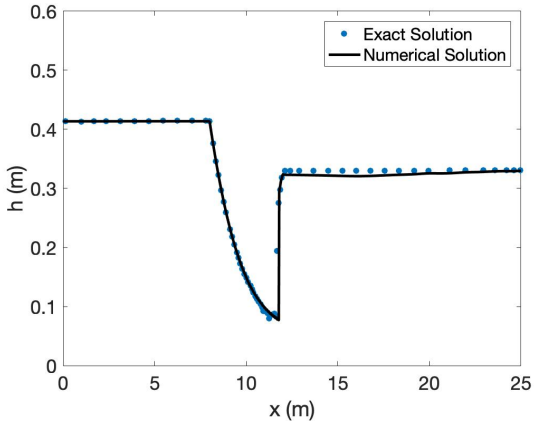
The steady-state solutions were obtained at time $T = 60\,sec$. As for the previous case, a CFL value of 1 was set. For this scenario, figure 6.10a shows that the predicted numerical water heights are in close agreement with the exact solution. Nevertheless, in figure 6.10b a slight deviation is seen in the case of the predicted discharge rates. Similar deviations were reported by several other studies [53, 60, 66].



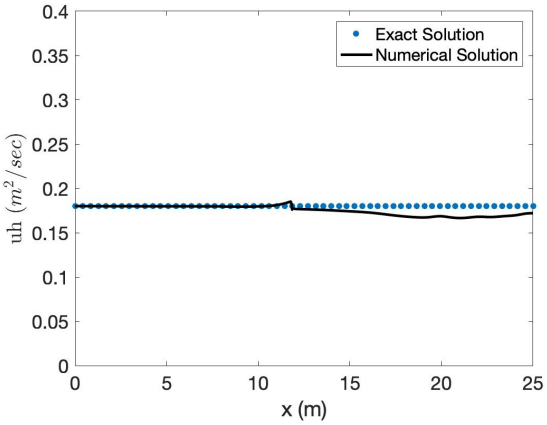

(a) Steady state solution of water height




(b) Steady state solution of discharge




(a) Steady state solution of h in a transcritical flow




(b) Steady state solution of discharge in a transcritical flow

### 6.2.4. Perturbation Over Elliptical Hump

The previous case can be extended in the two-dimensional space by introducing a bathymetry which is a function of x and y. Introduced by Leveque *et. al* [40], it consists of a $[0, 2] \times [0, 1]$ domain which is frictionless and completely wet. The topography is defined by

$$z(x, y) = 0.8e^{(-5(x-0.9)^2-50(y-0.5)^2)}$$

Therefore, the bathymetry is a elliptical hump as shown in figure 6.11. The initial conditions are given by

$$h = \begin{cases} 1.01, & \text{if } 0.05 < x < 0.15 \\ 1.0, & \text{if otherwise} \end{cases} \qquad uh = 0$$

$$vh = 0$$

Suppose subscripts b and i denote the boundary and inner cell values, then the boundary conditions are given by

1. *Transmissive condition*: Lateral Walls $(y = 0,\ y = 1)$ and Upstream $(x = 0)$

$$\{h\}_b = \{h\}_i$$
$$\{uh\}_b = \{uh\}_i$$
$$\{vh\}_b = \{vh\}_i$$

2. *Mirror condition*: Downstream $(x = 2)$

$$\{h\}_b = \{h\}_i$$
$$\{uh\}_b = -\{uh\}_i$$
$$\{vh\}_b = \{vh\}_i$$

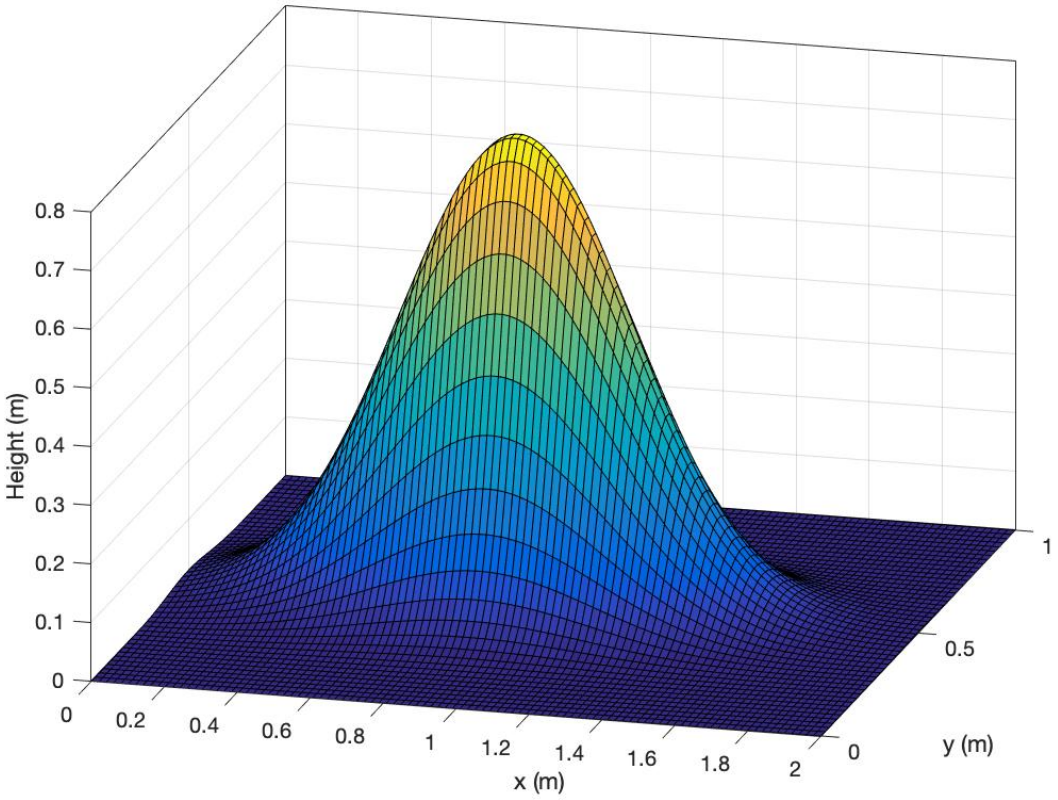



Figure 6.11: Elliptical profile of the bed

From the initial conditions, this scenario provides an ideal challenge to test the well-balanced nature of the model. Furthermore, the model can be tested for its capability to predict the multitude of complex features arising from the interactions between the waves and the varied topography of the bed. The test was conducted on a $300 \times 150$ fixed grid.

Figures 6.12 to 6.15 show the contours of the water heights at different times. At the start of the scenario, the surface perturbation instantly leads to the formation of 2 waves travelling in opposite directions. At t = 0.12 sec, the wave travelling west had already passed out of the domain. On the other hand, the wave travelling east gradually tends to distort due to spatially varying slopes of the topography. At t = 0.24 sec, a crest is formed as the perturbation passes over the hump. The energy of the gravity waves at the peak ultimately leads to it being dissipated through scattering as seen in figure 6.13. At t = 0.36 sec, one can see a multitude of interactions between the reflected and oncoming waves. The results obtained using the model are in general agreement with [44] and [40].


Figure 6.12: Water heights at $t = 0.12\,sec.$ Contour values range from $h = 0.9999\,m$ to $h = 1.0029\,m$

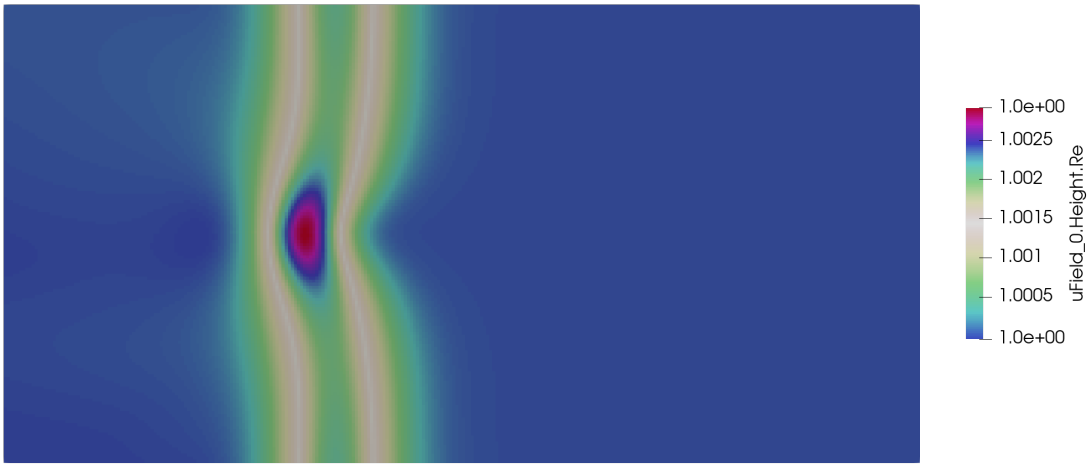



Figure 6.13: Water heights at $t = 0.24\,sec.$ Contour values range from $h = 0.9999\,m$ to $h = 1.0029\,m$

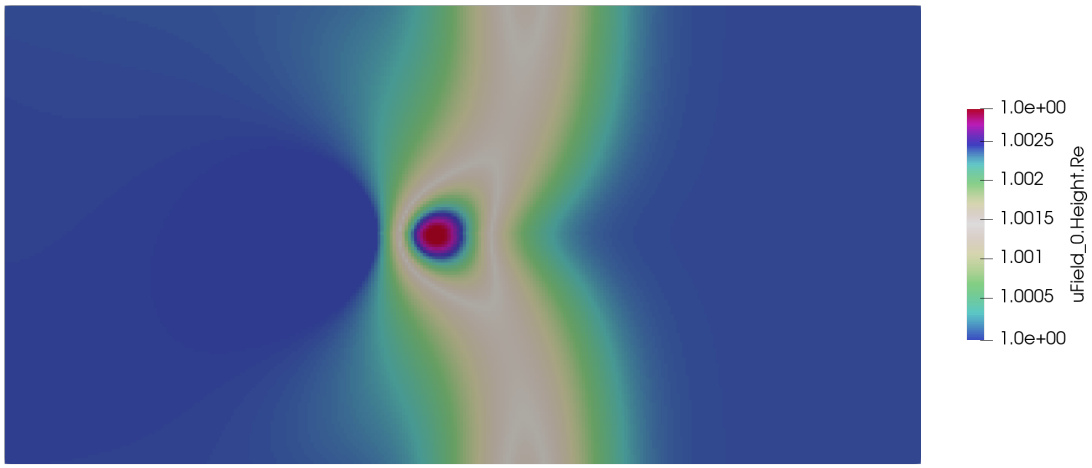



Figure 6.14: Water heights at $t = 0.36\,sec.$ Contour values range from $h = 0.99939\,m$ to $h = 1.0029\,m$

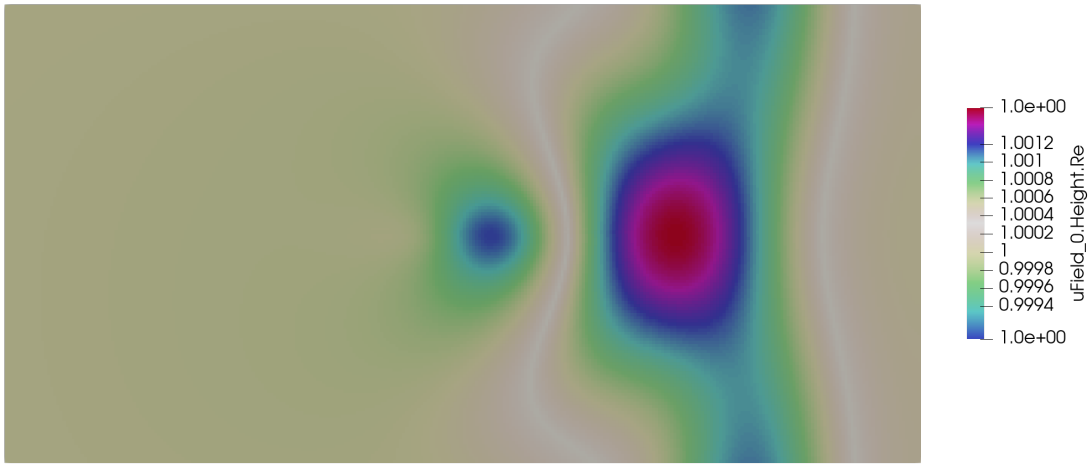



Figure 6.15: Water heights at $t = 0.48\,sec.$ Contour values range from $h = 0.99903\,m$ to $h = 1.0016\,m$

# 7

# Conclusion

The effect of the human footprint on the ecological balance is increasing as time passes. One of the disruptions caused is the rise in likelihood of devastating floods in urban environments. As a result, urban planners are persuaded to design and develop more robust critical infrastructure to best mitigate the damage caused by flash floods and storm surges. Numerical flood modelling provides a tool to aid in building smarter designs and thus help in reinforcing the critical infrastructure for longevity.

Due to the large scale nature of the domains, numerical solvers geared towards solving the flooding phenomena in urban landscapes must be highly efficient and accurate. This is achieved through the usage of time-adaptive grids which enables high-resolution solutions in various areas of interest and accurately predicting the wave propagation of the water. Thus the aim of the thesis is to build a scalable, and accurate hydrological solver capable of solving for time-adaptive grids. The hydrological solver must be able to handle highly non-uniform bathymetry and initial conditions as witnessed in many real-life scenarios. Furthermore, it was emphasized to base the solver on an open-source scientific library to promote its adoption and future development.

The goal is achieved by creating the h-refinement based hydrological model that utilizes the highly efficient and scalable numerical solvers offered by PETSc. The model is thus, able to solve on a quad-tree mesh generated and manipulated by the DMPlex and DMForest. Furthermore, a novel adaption strategy (section 5.5) is implemented to increase the adaptivity of the model to various scenarios. Using the proposed algorithms, various cases have been carried out as a step to validate the model and evaluate the performance of the model. Using the dam break scenarios (section 6.2.1) and the channel flow scenarios (section 6.2.3), it can be concluded the required accuracy and versatility of the model is demonstrated. Furthermore, the well-balanced nature of the fixed grid solver is verified in the perturbation over an elliptical hump (section 6.2.4).

### 7.0.1. Future Work

A number of improvements can be made to the model that would further elevate its scope of applications in the industry and academia. These are stated below:

- Performance: As shown in figures 6.3 and 6.7 obtained from the profiling studies, the parallel efficiencies of the fixed grid solvers can be considered to be lacking. Further optimizations can be made to the interface associated with the residual evaluation subroutines of the PETSc module. The same can be done for user-defined routines found within the written program such as in the Riemann solvers. This would also naturally lead to a boost in performance for the AMR solver. Next, the proposed adaption criteria subroutine (`adapttotolerance()`) can be further simplified and optimised.

- 'Well-Balancedness' of the AMR Solver: Although, the fixed grid solver was found to be well-balanced as demonstrated in section 6.2.4, the same cannot be applied to the AMR solver. Due to the difference in the DM types (Plex in the case of a fixed grid and P4est in the case of an adaptive grid), minor noise was found to be scattered across the domain. Upon further inspection, it was found that the problem most likely, lies in the PETSc-P4est interface as opposed to existing

in the implementation of the libraries themselves. Furthermore, the interface connecting the two libraries is comprehensive which prolongs debugging. As a result, it was decided to dig deeper into the issue and make the necessary corrections while the report is being evaluated. With an open conversation with the PETSc development team, the reality of this error correction is very much possible as was shown in the earlier stages of the project. Should there be a successful fix, the accuracy of the solutions obtained from the AMR solver can in-turn be increased.

- LIDAR Data: The cases presented in this project contain artificially generated bathymetry. However, in order to achieve the full potential of the model, a modular interface needs to be built to allow for the parsing of open-source LIDAR data of an area. This could ultimately allow solving for realistic flooding scenarios. Such a capability can be realised by making use of geospatial data processing libraries such as Shapely, Fiona, GeoPandas and/or GDAL to name a few. All of the stated libraries are open-source and can be incorporated into the model using Python wrappers provided by PETSc.

# Bibliography


[1] Shrirang Abhyankar, Jed Brown, Emil M Constantinescu, Debojyoti Ghosh, Barry F Smith, and Hong Zhang. Petsc/ts: A modern scalable ode/dae solver library. *arXiv preprint arXiv:1806.01437*, 2018.

[2] Martin Alnæs, Jan Blechta, Johan Hake, August Johansson, Benjamin Kehlet, Anders Logg, Chris Richardson, Johannes Ring, Marie E Rognes, and Garth N Wells. The fenics project version 1.5. *Archive of Numerical Software*, 3(100), 2015.

[3] Alexey Androsov, Jörn Behrens, and Sergey Danilov. Tsunami modelling with unstructured grids. interaction between tides and tsunami waves. In *Computational Science and High Performance Computing IV*, pages 191–206. Springer, 2011.

[4] Emmanuel Audusse, Christophe Chalons, and Philippe Ung. A simple well-balanced and positive numerical scheme for the shallow-water system. *Communications in Mathematical Sciences*, 13 (5):1317–1332, 2015.

[5] I Babuška and BQ Guo. The h-p version of the finite element method for domains with curved boundaries. *SIAM Journal on Numerical Analysis*, 25(4):837–861, 1988.

[6] Ivo Babuska, Barna A Szabo, and I Norman Katz. The p-version of the finite element method. *SIAM journal on numerical analysis*, 18(3):515–545, 1981.

[7] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[8] Satish Balay, Shrirang Abhyankar, Mark Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Alp Dener, Victor Eijkhout, W Gropp, et al. Petsc users manual. 2019.

[9] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Alp Dener, Victor Eijkhout, William D. Gropp, Dmitry Karpeyev, Dinesh Kaushik, Matthew G. Knepley, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Todd Munson, Karl Rupp, Patrick Sanan, Barry F. Smith, Stefano Zampini, Hong Zhang, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.13, Argonne National Laboratory, 2020. URL `https://www.mcs.anl.gov/petsc`.

[10] Jörn Behrens. Atmospheric and ocean modeling with an adaptive finite element solver for the shallow-water equations. *Applied Numerical Mathematics*, 26(1-2):217–226, 1998.

[11] Jörn Behrens, Natalja Rakowsky, Wolfgang Hiller, Dörthe Handorf, Matthias Läuter, Jürgen Päpke, and Klaus Dethloff. amatos: Parallel adaptive mesh generator for atmospheric and oceanic simulation. *Ocean modelling*, 10(1-2):171–183, 2005.

[12] Chris J Budd, Weizhang Huang, and Robert D Russell. Adaptivity with moving grids. *Acta Numerica*, 18:111–241, 2009.

[13] J Burguete, Pilar García-Navarro, and J Murillo. Friction term discretization and limitation to preserve stability and conservation in the 1d shallow-water model: Application to unsteady irrigation and river flow. *International Journal for Numerical Methods in Fluids*, 58(4):403–425, 2008.

[14] Carsten Burstedde, Lucas C Wilcox, and Omar Ghattas. p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, 33(3): 1103–1133, 2011.

[15] Xuan Cai, Holger Marschall, Martin Wörner, and Olaf Deutschmann. A phase field method with adaptive mesh refinement for numerical simulation of 3d wetting processes with openfoam®. In *2nd International Symposium on Multiscale Multiphase Process Engineering (MMPE), Hamburg, Germany. DECHEMA*, 2014.

[16] Vincenzo Casulli and Roy A Walters. An unstructured grid, three-dimensional model based on the shallow water equations. *International journal for numerical methods in fluids*, 32(3):331–348, 2000.

[17] Stephen P Cook. *Adaptive Mesh Methods for Numerical Weather Prediction*. PhD thesis, University of Bath, 2016.

[18] Stephen L Cornford, Daniel F Martin, Daniel T Graves, Douglas F Ranken, Anne M Le Brocq, Rupert M Gladstone, Antony J Payne, Esmond G Ng, and William H Lipscomb. Adaptive mesh, finite volume modeling of marine ice sheets. *Journal of Computational Physics*, 232(1):529–549, 2013.

[19] Sanderson L Gonzaga de Oliveira, Mauricio Kischinhevsk, and João Manuel RS Tavares. Novel graph-based adaptive triangular mesh refinement for finite-volume discretizations. 2013.

[20] Florence Drui, Alexandru Fikl, Pierre Kestener, Samuel Kokh, Adam Larat, Vincent Le Chenadec, and Marc Massot. Experimenting with the p4est library for amr simulations of two-phase flows. *ESAIM: Proceedings and Surveys*, 53:232–247, 2016.

[21] Alexandre Ern, Serge Piperno, and Karim Djadel. A well-balanced runge–kutta discontinuous galerkin method for the shallow-water equations with flooding and drying. *International journal for numerical methods in fluids*, 58(1):1–25, 2008.

[22] L Freret, L Ivan, Hans De Sterck, and Clinton PT Groth. High-order finite-volume method with block-based amr for magnetohydrodynamics flows. *Journal of Scientific Computing*, 79(1):176–208, 2019.

[23] Mikito Furuichi and Dave A May. Implicit solution of the material transport in stokes flow simulation: Toward thermal convection simulation surrounded by free surface. *Computer Physics Communications*, 192:1–11, 2015.

[24] Christophe Geuzaine and Jean-François Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities. *International journal for numerical methods in engineering*, 79(11):1309–1331, 2009.

[25] Babu Gheethaa. Well-balanced hydrological model based on 2d shallow water equations, 2019.

[26] Mukkund Sunjii Babu Gheethaa. Time-adaptive hydrological solver. `https://github.com/mukkund1996/petsc`, 2020.

[27] N Goutal. *Proceedings of the 2nd workshop on dam-break wave simulation*. Department Laboratoire National d'Hydraulique, Groupe Hydraulique Fluviale, 1997.

[28] Christopher J Greenshields. Openfoam user guide. *OpenFOAM Foundation Ltd, version*, 3(1): e2888, 2015.

[29] David A Ham, Julie Pietrzak, and Guus S Stelling. A scalable unstructured grid 3-dimensional finite volume model for the shallow water equations. *Ocean Modelling*, 10(1-2):153–169, 2005.

[30] Weizhang Huang and Robert D Russell. *Adaptive moving mesh methods*, volume 174. Springer Science & Business Media, 2010.

[31] Yuxin Huang, Ningchuan Zhang, and Yuguo Pei. Well-balanced finite volume scheme for shallow water flooding and drying over arbitrary topography. *Engineering Applications of Computational Fluid Mechanics*, 7(1):40–54, 2013.

[32] Tobin Isaac and Matthew G Knepley. Support for non-conformal meshes in petsc's dmplex interface. *arXiv preprint arXiv:1508.02470*, 2015.

[33] J Jansson, Ezhilmathi Krishnasamy, and M Leoni. Adaptive direct fem simulation with unicorn/fenics-hpc for cs1. 5th International Workshop on High Order CFD Methods, 2018.

[34] Hrvoje Jasak and AD Gosman. Automatic resolution control for the finite-volume method, part 2: Adaptive mesh refinement and coarsening. *Numerical Heat Transfer: Part B: Fundamentals*, 38 (3):257–271, 2000.

[35] George Karypis, Kirk Schloegel, and Vipin Kumar. Parmetis. *Parallel graph partitioning and sparse matrix ordering library. Version*, 2, 2003.

[36] Georges Kesserwani and Qiuhua Liang. Rkdg2 shallow-water solver on non-uniform grids with local time steps: Application to 1d and 2d hydrodynamics. *Applied Mathematical Modelling*, 39 (3-4):1317–1340, 2015.

[37] Ethan J Kubatko, Shintaro Bunya, Clint Dawson, and Joannes J Westerink. Dynamic p-adaptive runge–kutta discontinuous galerkin methods for the shallow water equations. *Computer Methods in Applied Mechanics and Engineering*, 198(21-26):1766–1774, 2009.

[38] Andrew B Lambe and Aleksander Czekanski. Topology optimization using a continuous density field and adaptive mesh refinement. *International Journal for Numerical Methods in Engineering*, 113(3):357–373, 2018.

[39] Michael Lange, Matthew G Knepley, and Gerard J Gorman. Flexible, scalable mesh and data management using petsc dmplex. *arXiv preprint arXiv:1505.04633*, 2015.

[40] Randall J LeVeque. Balancing source terms and flux gradients in high-resolution godunov methods: the quasi-steady wave-propagation algorithm. *Journal of computational physics*, 146(1): 346–365, 1998.

[41] Randall J LeVeque and David L George. High-resolution finite volume methods for the shallow water equations with bathymetry and dry states. In *Advanced numerical models for simulating tsunami waves and runup*, pages 43–73. World Scientific, 2008.

[42] Q Liang, AGL Borthwick, and G Stelling. Simulation of dam-and dyke-break hydrodynamics on dynamically adaptive quadtree grids. *International journal for numerical methods in fluids*, 46(2): 127–162, 2004.

[43] Qiuhua Liang. A structured but non-uniform cartesian grid-based model for the shallow water equations. *International Journal for Numerical Methods in Fluids*, 66(5):537–554, 2011.

[44] Qiuhua Liang and Alistair GL Borthwick. Adaptive quadtree simulation of shallow flows with wet–dry fronts over complex topography. *Computers & Fluids*, 38(2):221–234, 2009.

[45] Qiuhua Liang and Fabien Marche. Numerical resolution of well-balanced shallow water equations with complex source terms. *Advances in water resources*, 32(6):873–884, 2009.

[46] Kyle T Mandli and Clint N Dawson. Adaptive mesh refinement for storm surge. *Ocean Modelling*, 75:36–50, 2014.

[47] Tomislav Maric, Holger Marschall, and Dieter Bothe. vofoam-a geometrical volume of fluid algorithm on arbitrary unstructured meshes with local dynamic adaptive mesh refinement using openfoam. *arXiv preprint arXiv:1305.3417*, 2013.

[48] Tomislav Maric, Holger Marschall, and Dieter Bothe. vofoam-a geometrical volume of fluid algorithm on arbitrary unstructured meshes with local dynamic adaptive mesh refinement using openfoam. *arXiv preprint arXiv:1305.3417*, 2013.

[49] Andrew TT McRae, Colin J Cotter, and Chris J Budd. Optimal-transport–based mesh adaptivity on the plane and sphere using finite elements. *SIAM Journal on Scientific Computing*, 40(2): A1121–A1148, 2018.

[50] Daniel Rettenmaier, Daniel Deising, Yun Ouedraogo, Erion Gjonaj, Herbert De Gersem, Dieter Bothe, Cameron Tropea, and Holger Marschall. Load balanced 2d and 3d adaptive mesh refinement in openfoam. *SoftwareX*, 10:100317, 2019.

[51] Daniel Rettenmaier, Daniel Deising, Yun Ouedraogo, Erion Gjonaj, Herbert De Gersem, Dieter Bothe, Cameron Tropea, and Holger Marschall. Load balanced 2d and 3d adaptive mesh refinement in openfoam. *SoftwareX*, 10:100317, 2019.

[52] Jeremy A Riousset, Carol S Paty, Robert J Lillis, Matthew O Fillingim, Scott L England, Paul G Withers, and John P M Hale. Three-dimensional multifluid modeling of atmospheric electrodynamics in mars' dynamo region. *Journal of Geophysical Research: Space Physics*, 118(6):3647–3659, 2013.

[53] Benedict D Rogers, Alistair GL Borthwick, and Paul H Taylor. Mathematical balancing of flux gradient and source terms prior to using roe's approximate riemann solver. *Journal of Computational Physics*, 192(2):422–451, 2003.

[54] Eleuterio F Toro and Eleuterio Toro. *Shock-capturing methods for free-surface shallow flows*, volume 868. Wiley New York, 2001.

[55] Giovanni Tumolo, Luca Bonaventura, and Marco Restelli. A semi-implicit, semi-lagrangian, p-adaptive discontinuous galerkin method for the shallow water equations. *Journal of Computational Physics*, 232(1):46–67, 2013.

[56] Renato Vacondio, Alessandro Dal Palù, Alessia Ferrari, Paolo Mignosa, Francesca Aureli, and Susanna Dazzi. A non-uniform efficient grid type for gpu-parallel shallow water equations models. *Environmental Modelling & Software*, 88:119–137, 2017.

[57] Renato Vacondio, Alessandro Dal Palù, Alessia Ferrari, Paolo Mignosa, Francesca Aureli, and Susanna Dazzi. A non-uniform efficient grid type for gpu-parallel shallow water equations models. *Environmental Modelling & Software*, 88:119–137, 2017.

[58] Manuel Valera, Mary P. Thomas, Mariangel Garcia, and Jose E. Castillo. Parallel implementation of a PETSc-Based framework for the general curvilinear coastal ocean model. *Journal of Marine Science and Engineering*, 7(6), 2019. ISSN 2077-1312. doi: 10.3390/jmse7060185. URL https://www.mdpi.com/2077-1312/7/6/185.

[59] Alessandro Valiani and Lorenzo Begnudelli. Divergence form for bed slope source term in shallow water equations. *Journal of Hydraulic Engineering*, 132(7):652–665, 2006.

[60] Alessandro Valiani and Lorenzo Begnudelli. Divergence form for bed slope source term in shallow water equations. *Journal of Hydraulic Engineering*, 132(7):652–665, 2006.

[61] Emily Walsh. *Moving mesh methods for problems in meteorology*. PhD thesis, University of Bath, 2010.

[62] RA Walters and EJ Barragy. Comparison of h and p finite element approximations of the shallow water equations. *International journal for numerical methods in fluids*, 24(1):61–79, 1997.

[63] Yoshiki Yamazaki, Kwok Fai Cheung, and Zygmunt Kowalik. Depth-integrated, non-hydrostatic model with grid nesting for tsunami generation, propagation, and run-up. *International Journal for Numerical Methods in Fluids*, 67(12):2081–2107, 2011.

[64] Hong Zhang, Emil M. Constantinescu, and Barry F. Smith. PETSc TSAdjoint: a discrete adjoint ODE solver for first-order and second-order sensitivity analysis, journal = arXiv e-preprints, eprint = 1912.07696, archiveprefix = arXiv, year=2019.

[65] Yinglong J Zhang and Antonio M Baptista. An efficient and robust tsunami model on unstructured grids. part i: Inundation benchmarks. *Pure and Applied Geophysics*, 165(11-12):2229–2248, 2008.

[66] Jian G Zhou, Derek M Causon, Clive G Mingham, and David 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.