Al-Farabi Kazakh National University

UDC 004.94 On manuscript right

ARSHYN ALTYBAY

Development of high-performance parallel algorithms and software complex for modeling hyperbolic type equations with singular coefficients: tsunami and acoustic wave propagation

6D075100 – Computer Science, Computer Engineering and Management

Thesis for the Degree of
Doctor of Philosophy (PhD)

Scientific advisors:
Dr. Niyaz Tokmagambetov,
al-Farabi Kazakh National University

Prof. Michael Ruzhansky,
Ghent University, Belgium

Republic of Kazakhstan
Almaty, 2021

CONTENTS

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my supervisors Prof. Micheal Ruzhansky and Dr. Niyaz Tokmagambetov for the continuous support of my Ph.D study and research, for their motivation, enthusiasm, and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis.

Besides my supervisors, I would like to thank the rest of my thesis committee:

Prof. Baidaulet Urmashev, Dr. Timur Imankulov for their encouragement, insightful comments, and hard questions.

Also, my special thanks go to all members of Professor Micheal's "Ghent Analysis and PDE" research group for their motivation, enthusiasm, and assistance when we were working together in Ghent. I hope that further, we can work together not only as a mentor but also as a good friend.

And I would like to thank my colleagues from the department of computer science in KazNU Almas Temirbekov and Beimbet Daribayev for useful advice and my friends from the Institute of mathematics and computer modelling for motivation and encouragement.

Last but not the least; I would like to thank my big family: my mother Anigul Sayat, my father Altybay Dauytkazy, my brother Samgar, my sister Dina and my wife Hafiza Jumay. I believe that from their support, I can finish my Ph.D study on time.

ABSTRACT

This thesis is devoted to the numerical approximation and parallel computing of the hyperbolic type equations with singular coefficients such tsunami and acoustic wave equations, and development of software complex for investigation of wave equation with singular coefficients.

In the first chapter, we present the mathematical models of hyperbolic type equations with singular coefficients such as tsunami and acoustic wave equation. Then we transform the partial differential equations to finite difference schemes. We compare explicit and implicit finite difference schemes, as a result of comparing we choose implicit schemes for our further implementation.

Our numerical realization based on the implicit finite difference scheme and the robustness of this scheme is validated on the tsunami wave equations model. And in this section we have briefly described the concept of very weak solutions.

At the end of this chapter we used our model to study the Caspian tsunami and, then we carried out numerical modeling and made different predictions relative to the tsunami, reaching the shore, depending on the height of the initial wave.

In the second chapter, we consider parallel numerical implementation hyperbolic type equations with singular coefficients. Firstly, we present MPI implementation of 2D wave equation with a distributional coefficient then CUDA implementation of 2D tsunami wave equation and related computational results.

After we present a hybrid implementation of acoustic wave problems, then we compare the results from the different implementations.

In a hybrid implementation, the joint use of OpenMP, CUDA and MPI technologies to solve one problem, the result of the calculations showed that implementation of this gives good results.

In the third chapter, we describe the software complex which we developed for investigation of wave equation with singular coefficients. This software is open-source, cross-platform, and written in, one of the modern programming languages Python. This software will help researchers who investigate hyperbolic systems with singular coefficients.

Keywords: *tsunami wave equation, implicit difference scheme, parallel computing, MPI, CUDA, numerical simulation.*

INTRODUCTION

This thesis is devoted to the development of high-performance parallel algorithms and software complex for modeling tsunami and acoustic wave equations with irregular coefficients.

Relevance of the research topic. At present, randomly occurring and rapidly changing processes leads to huge environmental and economic problems. Therefore, the modeling of such processes is very important. Many such problems are modeled by hyperbolic type equations with singular coefficients.

We are allowing h to be a (positive) distribution, for example, allowing the case $h = 1 + \delta$, involving the δ-distribution. Such type of setting appears in applications, for example when one is looking at the behaviour of a particle in irregular electromagnetic fields: in the case of Landau Hamiltonian on R^n, and the corresponding wave equation was analysed by the authors in [6]. While from the physical point of view (of irregular electromagnetic fields) such situation is natural and one expects the well-posedness, mathematically the equation is difficult to handle because of the general impossibility to multiply distributions (recall the famous Schwartz impossibility result from [80]).

Here if the singular coefficients are delta like function, then there is no classical solution. To deal with such problems we use the concept of very weak solutions [5-8].

The simulation of physical processes mentioned above on a large scale and for a long time requires large computational costs. If the computational algorithm is sequential, then the computational costs are even larger. A temporary solution to avoid this problem is parallelization.

Efficiently parallelizing numerical methods and algorithms on a multicore processor was born in 2004 due to the fact that the physical limit forced the use of more processors on a silicon crystal.

Many engineering and scientific applications often require the simultaneous solution of a large number of equations with variable coefficients. The primary aim of this thesis work is to take advantage of the computational power of various modern parallel processor architectures to accelerate the computational speed of some mathematical problems by giving new algorithms and solutions.

The purpose of the dissertation work. Development of high-performance parallel algorithms and software complex for numerical solutions of hyperbolic type equations with singular coefficients such as tsunami and acoustic wave equation.

Research objectives realizing the goal of the dissertation work:

1) To design and analyze finite difference schemes for the 1D and 2D hyperbolic type equations, and to elaborate and study the implicit finite difference scheme for our equations;

2) To solve numerically the tsunami and acoustic wave equation in one and two dimensions by using the implicit finite difference scheme;

3) To parallelize the sequential algorithms using CUDA and MPI technologies;

4) To develop software complex for the investigation of the wave equation with singular coefficients.

The object of study. High-performance parallel computing, numerical methods, parallel programming technologies, hyperbolic type partial differential equations with singular coefficients, finite difference schemes, software application design tools.

The subject of study. Numerical analysis, numerical methods, parallel numerical algorithms for solving tridiagonal systems, software development technologies.

Scientific novelty. Proof of the existence, uniqueness and consistency of very weak solutions of the tsunami equation and justification by numerical modeling.

Development of a parallel algorithm for the numerical solution of the two-dimensional wave equation with a singular coefficient using the MPI technology based on an implicit difference scheme.

Development of a parallel algorithm for the numerical solution of the two-dimensional tsunami equation using the CUDA technology based on an implicit difference scheme.

Development of a parallel hybrid algorithm for the numerical solution of the two-dimensional acoustics wave equation based on an implicit difference scheme.

Development of open-source, cross-platform software complex for numerical solution and investigation of hyperbolic type equation with singular coefficients.

The main provision for the defense

- Proof of the existence, uniqueness and consistency of very weak solutions of the tsunami equation and justification by numerical simulations.
- The developed parallel computational algorithm for the numerical solution of the two-dimensional wave equation with singular coefficients.
- The developed parallel algorithm for the numerical solution of the two-dimensional tsunami equation using the CUDA technology.
- The developed parallel hybrid algorithm for the numerical solution of the two-dimensional acoustics wave equation.
- The developed software complex for investigation of hyperbolic type equations with singular coefficients.

The theoretical significance of this work lies on the existence, uniqueness and consistency of very weak solutions to the tsunami equation and justified by numerical simulations.

The practical significance of the work is as follows:

The developed parallel algorithms for the numerical solution of hyperbolic equations with singular coefficients are applied to simulate a tsunami in the Caspian Sea; Developed software can be used to study waves in heterogeneous media in various fields of science.

Volume and structure of work. The thesis consists of an introduction, 3 sections and a conclusion, a list of references and an appendix. The total volume of the thesis is 100 pages, 40 figures, 8 tables. The list of references consists of 79 titles.

In the introduction, the relevance of the topic of the dissertation work, goals, as well as tasks for achieving this goal are discussed. The results obtained so far, their scientific novelty and significance are described.

In the first chapter, we present the mathematical models of hyperbolic type equations with singular coefficients such as tsunami and acoustic wave equation. Then we transform the partial differential equations to finite difference schemes. We compare explicit and implicit finite difference schemes, as a result of comparing we choose implicit schemes for our further implementation.

At the end of this chapter we use our model to study the Caspian tsunami then we carried out numerical modeling and make different predictions related to the tsunami, reaching the shore, depending on the height of the initial wave.

In the second chapter, we consider parallel numerical implementation of hyperbolic type wave equations. Firstly, we present MPI implementation of 2D wave equation with a distributional coefficient then CUDA implementation of 2D tsunami wave equation and related computational results.

At the end of this chapter, we present a hybrid implementation of acoustic wave problem then we compare the results of the different implementations.

In a hybrid implementation, joint use of OpenMP, CUDA and MPI technologies to solve one problem, the result of the calculations shows that this implementation gives very good results.

In the third chapter, we describe the software complex for investigation of wave equation with singular coefficients. This software is open-source, cross-platform, and written in, one of the modern programming languages Python. This software will help researchers who investigate hyperbolic type equations with singular coefficients.

In the conclusion, conclusions of this dissertation work are presented.

Publication. On the topic of the dissertation, 9 papers have been published, including 4 in publications recommended by the Committee for Control in the Sphere of Education and Science of the Ministry of Education and Science of the Republic of Kazakhstan, 2 works in peer-reviewed journals included in the international citation base SCOPUS, 3 works in collections of international conferences.

1. Altybay A., Ruzhansky M., Tokmagambetov N. Wave equation with distributional propagation speed and mass term: numerical simulations // Appl. Math. E-Notes. – 2019. – Vol. 19. – P. 552-562. (Scopus Q3).
2. Altybay A., Ruzhansky M., Tokmagambetov N. A parallel hybrid implementation of the 2D acoustic wave equation // International Journal of Nonlinear Sciences and Numerical Simulation. – 2020. – Vol. 21, Iss. 7-8. – P. 821-827. (Scopus, Q2).

3. 11 Altybay A., Ruzhansky M., Sebih M. E., Tokmagambetov N. Fractional Klein-Gordon equation with strongly singular mass term // Chaos, Solitons & Fractals. – 2021. –Vol. 143. – P. 110579-110647(Scopus, Q1).
4. 12 Altybay A., Ruzhansky M., Sebih M. E., Tokmagambetov N. Fractional Schr¨odingerEquations with potentials of higher-order singularities // Reports on Mathematical Physics. – 2021. – Vol. 87. №1. – P. 129-144(Scopus, Q3).
5. 13 Altybay A., Ruzhansky M., Sebih M. E., Tokmagambetov N. The heat equation with singular potentials // Applied Mathematics and Computation. – 2021. – Vol. 399. – P. 126-132(Scopus, Q1).
6. Altybay A., Tokmagambetov N. On numerical simulations of the 1D wave equation with a distributional coefficient and sourse term // International Journal of Mathematics and Physics. Al-Farabi Kazakh National University. – 2017. – Vol. 8. №2. – P. 28-33.
7. Altybay A., Tokmagambetov N. A parallel algorithm for solving the two-dimensional wave equation with a singular coefficient // KazNTU Bulletin. – 2019. – Vol. 1. – P. 404-410.
8. Altybay A., Tokmagambetov N. MPI parallel implement of a wave equation using an implicit finite difference scheme. // KBTU Bulletin. – 2020. №1(52). – P. 112-120.
9. Altybay A., Tokmagambetov N. GPU computing for 2d wave equation based on implicit finite difference schemes // Bulletin NIA RK. – 2020. №3(77). – P. 32-42.

1 MATHEMATICAL MODELS AND FINITE DIFFERENCE SCHEMES OF TSUNAMI AND ACOUSTIC WAVE EQUATION

1.1 Physical background of Tsunami and acoustic wave equations

Physical background of tsunami

"Tsunami" comes from two Japanese words "tsu" (harbor, port) and "nami" (wave). A tsunami is a huge wave caused by an earthquake on the seabed or a volcanic eruption, displacement of the underwater crust. Tsunamis are often caused by underwater earthquakes and are one of the most destructive natural forces in the world. Its speed is the same as that of an airplane, destroying coastal continents and settlements.

An earthquake generates a tsunami if it is of sufficient force and there is violent movement of the earth to cause substantial and sudden displacement of a massive amount of water. Indeed, a tsunami is not a single wave but a series of waves, also known as a wave train. The first wave in a tsunami is not necessarily the most destructive. Tsunamis are not tidal waves.

When a tsunami occurs in the deep sea, its height is about 1 m, and the propagation speed is very high, when it reaches the shallow sea, the speed begins to grow slowly, the wave height can reach a height of 35 m. Surprisingly, tsunami waves can be very long (as much as 60 miles, or 100 kilometers) and be as far as one hour apart. They are able to cross entire oceans without great loss of energy. The Indian Ocean tsunami traveled as much as 5,000 kilometers to Africa, arriving with sufficient force to kill people and destroy property.

Scientists say that a great earthquake of magnitude 9 struck the Pacific Northwest in 1700 and created a tsunami that caused flooding and damage on the Pacific coast of Japan.

If we consider the worst tsunami in recent history the hardest event occurred on December 26, 2004, a magnitude 9.1 earthquake struck northern Indonesia, affecting 14 countries in the Indian Ocean and killing about 230,000 people. July 17, 2006, a magnitude 7.7 earthquake struck off the town of Pangandaran and set off a tsunami of 2 m high which had killed more than 300 people. March 11, 2011, a magnitude 9.0 earthquake near Tōhoku caused a great tsunami struck and killed more than 18000 people.

In the Caspian Sea, earthquakes have been very frequent in recent years, for example, 10 earthquakes in 2016, 12 earthquakes in 2017, the last earthquake in February 2020. According to historical data, in 957 an earthquake shook the Caspian Sea and destroyed 15 settlements in the Iranian region. On May 14, 1970, as a result of an earthquake in the Buinak region near the epicenter of Makhachkala, a tsunami occurred in the Caspian Sea, as a result of which 20 settlements were flooded and areas of Makhachkala were flooded, 31 people died and 45,000 were left homeless. At the beginning of 2000, the Russian Institute of Oceanology published a brochure entitled

"Tsunamis in the Caspian Sea", which predicts a possible tsunami wave height of 3 meters in the Caspian and Black Seas. Therefore, modeling and forecasting tsunami waves on the coast is very important.

Mathematical model of tsunami propagation

In general, the tsunami waves can be modeled by shallow water equations, Boussinesq-type Equations, and Computational Fluid Dynamics.

The shallow water equations (SWE) are one of the tsunami propagation models. When the wavelength is much longer than the depth of water, the equations describe tsunami wave propagations.

The tsunami wave equation is one of the fundamental equations in many engineering and physical sciences. So one makes predictions and simulations of the practical interest. In this chapter we consider mathematical models and numerical simulations of the Cauchy problem for the tsunami propagation.

Governing Equation. The fundamental equations of tsunami propagation in the shallow water are given by Imamura and et.al [1].

$$\frac{\partial \eta}{\partial t} + \frac{\partial M}{\partial x} + \frac{\partial N}{\partial y} = 0,$$

$$\frac{\partial M}{\partial t} + \frac{\partial}{\partial x}\left(\frac{M^2}{D}\right) + \frac{\partial}{\partial y}\left(\frac{MN}{D}\right) + gD\frac{\partial \eta}{\partial x} + \frac{gn^2}{D^{7/3}} M\sqrt{M^2+N^2} = 0\,,$$

$$\frac{\partial N}{\partial t} + \frac{\partial}{\partial x}\left(\frac{MN}{D}\right) + \frac{\partial}{\partial y}\left(\frac{N^2}{D}\right) + gD\frac{\partial \eta}{\partial y} + \frac{gn^2}{D^{7/3}} N\sqrt{M^2+N^2} = 0, \qquad (1.1)$$

Where η is water surface elevation, h is the depth of the water, $D = h(x,y) + \eta$ is the total water depth, M and N are discharge fluxes in the x and y directions, g is the gravitational constant, n is a coefficient of bottom friction.

The 1D case. As the first step we describe the numerical scheme of the tsunami model, for linearized long wave equation without bottom friction in one dimensional propagation, given by Eq.(1.2)

$$\frac{\partial \eta}{\partial t} + \frac{\partial M}{\partial x} = 0$$

$$\frac{\partial M}{\partial t} + gD\frac{\partial \eta}{\partial x} = 0. \qquad (1.2)$$

Equation (1.2) are manifestations of mass and momentum conservation law, respectively. Frictional terms have been ignored in our discussion of SWE.

This system can namely be reduced to one equation for the sea level,

$$\frac{\partial^2 u(t,x)}{\partial t^2} = \frac{\partial}{\partial x}\left(H(t,x)\frac{\partial u(t,x)}{\partial x}\right) + f(t,x). \qquad (1.3)$$

In fact, 90% of the world's tsunamis are caused by underwater earthquakes [2] so if we want to allow a moving bottom in the model we can write tsunami model as follows[3]:

$$\frac{\partial^2 u(t,x)}{\partial t^2} = \frac{\partial}{\partial x}\left(H(t,x)\frac{\partial u(t,x)}{\partial x}\right) + \frac{\partial^2 H(t,x)}{\partial t^2} + f(t,x). \quad (1.4)$$

The t-dependence in H gives a moving bottom to model, such as an under-water slide or earthquake. The above equation can be written as the following:

$$\begin{cases} u_{tt}(t,x) - \partial_x\left(H(t,x)\partial_x u(t,x)\right) + \partial_{tt}H(t,x) = f(t,x), (\mathrm{t},\mathrm{x}) \in (0,\mathrm{T}) \times \Omega, \\ \mathrm{u}(0,\mathrm{x}) = \varphi(\mathrm{x}), u_t(0,x) = \psi(x), (\mathrm{x}) \in \Omega, \\ \mathrm{u}(\mathrm{t},\mathrm{x})|\,\partial\Omega = \mathrm{g}(\mathrm{t},\mathrm{x}), \mathrm{t} \in [0,\mathrm{T}], \end{cases} \quad (1.5)$$

where $\Omega = (0, X)$ for some fixed $X > 0$.

Where u represents the free surface displacement, and $H(t,x)$ is the still-water depth (typically obtained from a bathymetric map).

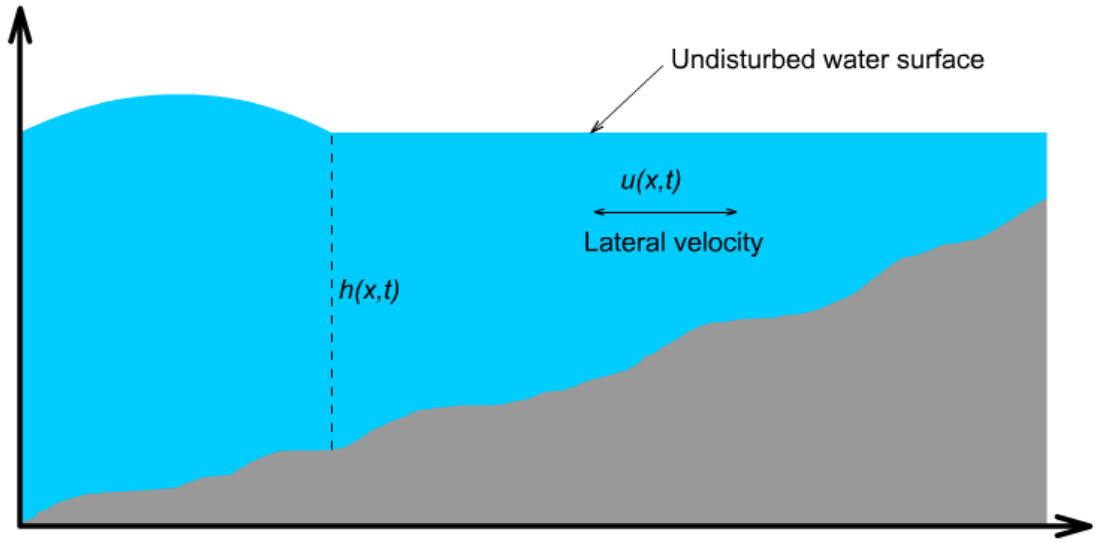


Figure 1.1 – Water level profile of an example 1D case

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial x}\left(h(x)\frac{\partial u}{\partial x}\right)$$

The term is common for many models of physical Phenomena

1.2 Very weak solutions to tsunami equation

In this context the concept of very weak solutions introduced in [4], for the analysis of second order hyperbolic equations with irregular coefficients, was applied in a series of papers [5], [6] and [7] for different physical models, in order to show a wide applicability. In [8, 9] it was applied for a damped wave equation with irregular dissipation arising from acoustic problems and an interesting phenomenon of the reflection of the original propagating wave was numerically observed. In all these papers the theory of very weak solutions is dealt for time-dependent equations.

In the recent works [10, 11, 12, 13], the authors start to study the concept of very weak solutions for partial differential equations with space-depending coefficients [81].

2.2.1 Existence of a very weak solution. In what follows, we consider the Cauchy problem

$$\begin{cases} u_{tt}(t,x) - \sum_{j=1}^{d} \partial_{x_j}\left(h_j(x)\partial_{x_j}u(t,x)\right) = 0, \ \ (t,x) \in [0,T] \times \mathbb{R}^d, \\ u(0,x) = u_0(x), \ \ u_t(0,x) = u_1(x), \ \ x \in \mathbb{R}^d, \end{cases} \tag{1.12}$$

with singular coefficients and initial data. Now we want to prove that it has a very weak solution. To start with, we regularise the coefficients h_i and the Cauchy data u_0 and u_1 by convolution with a suitable mollifier ψ, generating families of smooth functions $(h_{i,\varepsilon})_\varepsilon$, $(u_{0,\varepsilon})_\varepsilon$ and $(u_{1,\varepsilon})_\varepsilon$, that is

$$h_{i,\varepsilon}(x) = h_i * \psi_\varepsilon(x) \ \ for \ \ i = 1, \dots, d \tag{1.13}$$

and

$$u_{0,\varepsilon}(x) = u_0 * \psi_\varepsilon(x), \ \ u_{1,\varepsilon}(x) = u_1 * \psi_\varepsilon(x), \tag{1.14}$$

where

$$\psi_\varepsilon(x) = \varepsilon^{-1}\psi(x/\varepsilon), \ \ \varepsilon \in (0,1]. \tag{1.15}$$

The function ψ is a Friedrichs-mollifier, i.e. $\psi \in C_0^\infty(R^d)$, $\psi \geq 0$ and $\int \psi = 1$.

Assumption 1.1 *In order to prove the well posedness of the Cauchy problem (1.12) in the very weak sense, we ask for the regularisations of the coefficients $(h_{i,\varepsilon})_\varepsilon$ and the Cauchy data $(u_{0,\varepsilon})_\varepsilon$, $(u_{1,\varepsilon})_\varepsilon$ to satisfy the assumptions that there exist $N_0, N_1, N_2 \in N_0$ such that*

$$\| h_{i,\varepsilon} \|_{W^{1,\infty}} \lesssim \varepsilon^{-N_0}, \tag{1.16}$$

for $i = 1, \dots, d$ and

$$\| u_{0,\varepsilon} \|_{H^2} \lesssim \varepsilon^{-N_1}, \quad \| u_{1,\varepsilon} \|_{H^1} \lesssim \varepsilon^{-N_2}. \tag{1.17}$$

Remark 1.1 *We note that making an assumption on the regularisation is more general than making it on the function itself. We also mention that such assumptions on distributional coefficients, are natural. Indeed, we know that for $T \in E'(R^d)$ we can find $n \in N$ and functions $f_\alpha \in C(R^d)$ such that, $T = \sum_{|\alpha| \leq n} \partial^\alpha f_\alpha$. The convolution of T with a mollifier gives*

$$T * \psi_\varepsilon = \sum_{|\alpha| \leq n} \partial^\alpha f_\alpha * \psi_\varepsilon = \sum_{|\alpha| \leq n} f_\alpha * \partial^\alpha \psi_\varepsilon = \sum_{|\alpha| \leq n} \varepsilon^{-|\alpha|} f_\alpha * (\varepsilon^{-1}\partial^\alpha \psi(x/\varepsilon)), \tag{1.18}$$

and we easily see that the regularisation of T satisfy the above assumption. Fore more details, we refer to the structure theorems for distributions (see, e.g. [14]).

Definition 1 (Moderateness)

i. A net of functions $(f_\varepsilon)_\varepsilon$, is said to be H^1-moderate, if there exist $N \in N_0$ such that
$$\| g_\varepsilon \|_{H^1} \lesssim \varepsilon^{-N}.$$
ii. A net of functions $(g_\varepsilon)_\varepsilon$, is said to be H^2-moderate, if there exist $N \in N_0$ such that
$$\| g_\varepsilon \|_{H^2} \lesssim \varepsilon^{-N}.$$

iii. A net of functions $(h_\varepsilon)_\varepsilon$, is said to be $W^{1,\infty}$-moderate, if there exist $N \in N_0$ such that
$$\| h_\varepsilon \|_{W^{1,\infty}} \lesssim \varepsilon^{-N}.$$

iv. A net of functions $(u_\varepsilon)_\varepsilon$ from $C([0,T];H^2(R^d)) \cap C^1([0,T];H^1(R^d))$ is said to be C^1-moderate, if there exist $N \in N_0$ such that
$$\| u_\varepsilon(t,\cdot) \| \lesssim \varepsilon^{-N}$$

for all $t \in [0,T]$.

We note that if $h_i \in E'(R^d)$ for $i = 1,\dots d$ and $u_0, u_1 \in E'(R^d)$, then the regularisations $(h_{i,\varepsilon})_\varepsilon$ for $i = 1,\dots d$ of the coefficients and $(u_{0,\varepsilon})_\varepsilon$, $(u_{1,\varepsilon})_\varepsilon$ of the Cauchy data, are moderate in the sense of the last definition.

Definition 1 (Very weak solution) *The net* $(u_\varepsilon)_\varepsilon \in C([0,T];H^1(R^d)) \cap C^1([0,T];L^2(R^d))$ *is said to be a very weak solution to the Cauchy problem (1.12), if there exist*

$-W^{1,\infty}$-moderate regularisations of the coefficients h_i, for $i = 1,\dots d$,

$-H^2$-moderate regularisation of u_0,

$-H^1$-moderate regularisation of u_1,

such that $(u_\varepsilon)_\varepsilon$ solves the regularised problem

$$\{\partial_t^2 u_\varepsilon(t,x) - \sum_{j=1}^d \partial_{x_j}\left(h_{j,\varepsilon}(x)\partial_{x_j}u_\varepsilon(t,x)\right) = 0, \ \ (t,x) \in [0,T]\times R^d, u_\varepsilon(0,x) = u_{0,\varepsilon}(x), \ \ \partial_t u_\varepsilon(0,x) = u_{1,\varepsilon}(x), \ \ x \in R^d, \quad (1.19)$$

for all $\varepsilon \in (0,1]$, and is C^1-moderate.

Theorem 1.1 (Existence) *Let the coefficients* (h_i) *be positive in the sense that all regularisations* $(h_i)_\varepsilon$ *are positive, for* $i = 1,\dots,d$, *and assume that the regularisations of* h_i, u_0, u_1 *satisfy the assumptions (1.16) and (1.17). Then the Cauchy problem (1.12) has a very weak solution.*

Proof. The nets $(h_{i,\varepsilon})_\varepsilon$, for $i = 1,\dots,d$ and $(u_{0,\varepsilon})_\varepsilon$, $(u_{1,\varepsilon})_\varepsilon$ are moderate by assumption. To prove the existence of a very weak solution, it remains to prove that the net $(u_\varepsilon)_\varepsilon$, solution to the regularised Cauchy problem (1.22), is C^1-moderate. Using the estimates (2.2), (2.3)[81] and the moderateness assumptions (1.16) and (1.17), we arrive at
$$\| u_\varepsilon(t,\cdot) \| \lesssim \varepsilon^{-2N_0 - max\{N_1,N_2\}},$$
for all $t \in [0,T]$. This concludes the proof.

In the next sections, we want to prove the uniqueness of the very weak solution to the Cauchy problem (1.12) and its consistency with the classical solution when the latter exists.

Uniqueness. Let us assume that we are in the case when very weak solutions to the Cauchy problem (1.15) exist.

Definition 2 (Uniqueness) *We say that the Cauchy problem (*1.12*), has a unique very weak solution, if for all families of regularisations* $(h_{i,\varepsilon})_\varepsilon$, $(\tilde{h}_{i\varepsilon})_\varepsilon$, $(u_{0,\varepsilon})_\varepsilon$, $(\tilde{u}_{0,\varepsilon})_\varepsilon$

and $(u_{1,\varepsilon})_\varepsilon$, $(\tilde{u}_{1,\varepsilon})_\varepsilon$ *of the coefficients* h_i, *for* $i = 1, \dots d$ *and the Cauchy data* u_0, u_1, *satisfying*

$$\| h_{i,\varepsilon} - \tilde{h}_{i,\varepsilon} \|_{W^{1,\infty}} \leq C_k \varepsilon^k \ \ forall \ \ k > 0,$$
$$\| u_{0,\varepsilon} - \tilde{u}_{0,\varepsilon} \|_{H^1} \leq C_m \varepsilon^m \ \ forall \ \ m > 0,$$

and

$$\| u_{1,\varepsilon} - \tilde{u}_{1,\varepsilon} \|_{L^2} \leq C_n \varepsilon^n \ \ forall \ \ n > 0,$$

we have

$$\| u_\varepsilon(t,\cdot) - \tilde{u}_\varepsilon(t,\cdot) \|_{L^2} \leq C_N \varepsilon^N,$$

for all $N > 0$, where $(u_\varepsilon)_\varepsilon$ and $(\tilde{u}_\varepsilon)_\varepsilon$ are the families of solutions to the related regularised Cauchy problems.

Theorem 1.2 (Uniqueness) *Let* $T > 0$. *Suppose that* $h_i(x) = h(x)$ *for all* $i = 1, \dots, d$. *Assume that for* $i = 1, \dots d$, *the regularisations of the coefficients* h_i *and the regularisations of the Cauchy data* u_0 *and* u_1 *satisfy the assumptions (1.16) and (1.17). Then, the very weak solution to the Cauchy problem (*1.12*) is unique.*

Proof. Let $(h_{i,\varepsilon}, u_{0,\varepsilon}, u_{1,\varepsilon})_\varepsilon$, $(\tilde{h}_{i\varepsilon}, \tilde{u}_{0,\varepsilon}, \tilde{u}_{1,\varepsilon})_\varepsilon$ be regularisations of the coefficients h_i, for $i = 1, \dots d$ and the Cauchy data u_0, u_1, and let assume that they satisfy $\| h_{i,\varepsilon} - \tilde{h}_{i,\varepsilon} \|_{W^{1,\infty}} \leq C_k \varepsilon^k \ \ forall \ \ k > 0,$

$$\| u_{0,\varepsilon} - \tilde{u}_{0,\varepsilon} \|_{H^1} \leq C_m \varepsilon^m \ \ forall \ \ m > 0,$$

and

$$\| u_{1,\varepsilon} - \tilde{u}_{1,\varepsilon} \|_{L^2} \leq C_n \varepsilon^n \ \ forall \ \ n > 0.$$

Let us denote by $U_\varepsilon(t,x) := u_\varepsilon(t,x) - \tilde{u}_\varepsilon(t,x)$, where $(u_\varepsilon)_\varepsilon$ and $(\tilde{u}_\varepsilon)_\varepsilon$ are the solutions to the families of regularised Cauchy problems, related to the families $(h_{i,\varepsilon}, u_{0,\varepsilon}, u_{1,\varepsilon})_\varepsilon$ and $(\tilde{h}_{i\varepsilon}, \tilde{u}_{0,\varepsilon}, \tilde{u}_{1,\varepsilon})_\varepsilon$. Easy calculations show that U_ε solves the Cauchy problem

$$\{\partial_t^2 U_\varepsilon(t,x) - \sum_{j=1}^d \partial_{x_j}\left(\tilde{h}_{j,\varepsilon}(x)\partial_{x_j} U_\varepsilon(t,x)\right) = f_\varepsilon(t,x), \ \ (t,x) \in [0,T] \times R^d, U_\varepsilon(0,x) = (u_{0,\varepsilon} - \tilde{u}_{0,\varepsilon})(x), \ \ \partial_t U_\varepsilon(0,x) = (u_{1,\varepsilon} - \tilde{u}_{1,\varepsilon})(x), \ \ x \in R^d, \quad (1.20)$$

where

$$f_\varepsilon(t,x) = \sum_{j=1}^d \partial_{x_j}\left[\left(h_{j,\varepsilon}(x) - \tilde{h}_{j,\varepsilon}(x)\right)\partial_{x_j} u_\varepsilon(t,x)\right]. \quad (1.21)$$

By Duhamel's principle (see, e.g. [15]), we obtain the following representation

$$U_\varepsilon(t,x) = V_\varepsilon(t,x) + \int_0^t W_\varepsilon(x, t-s; s)ds, \quad (1.22)$$

for U_ε, where $V_\varepsilon(t,x)$ is the solution to the homogeneous problem

$$\{\partial_t^2 V_\varepsilon(t,x) - \sum_{j=1}^d \partial_{x_j}\left(\tilde{h}_{j,\varepsilon}(x)\partial_{x_j} V_\varepsilon(t,x)\right) = 0, \ \ (t,x) \in [0,T] \times R^d, V_\varepsilon(0,x) = (u_{0,\varepsilon} - \tilde{u}_{0,\varepsilon})(x), \ \ \partial_t V_\varepsilon(0,x) = (u_{1,\varepsilon} - \tilde{u}_{1,\varepsilon})(x), \ \ x \in R^d, \quad (1.23)$$

and $W_\varepsilon(x,t;s)$ solves

$$\{\partial_t^2 W_\varepsilon(x,t;s) - \sum_{j=1}^d \partial_{x_j}\left(\tilde{h}_{j,\varepsilon}(x)\partial_{x_j} W_\varepsilon(x,t;s)\right) = 0, \ \ (t,x) \in [0,T] \times R^d, W_\varepsilon(x,0;s) = 0, \ \ \partial_t W_\varepsilon(x,0;s) = f_\varepsilon(s,x), \ \ x \in R^d. \quad (1.24)$$

Taking the L^2 norm on both sides in (1.23) and to estimate V_ε and W_ε, we obtain

$$\| U_\varepsilon(\cdot,t) \|_{L^2} \leq \| V_\varepsilon(\cdot,t) \|_{L^2} + \int_0^T \| W_\varepsilon(\cdot, t-s; s) \|_{L^2} \, ds$$

$$\lesssim \left(1+\sum_{j=1}^{d} \| \tilde{h}_{j,\varepsilon} \|_{L^\infty}^{\frac{1}{2}}\right)\Big[\| u_{0,\varepsilon}-\tilde{u}_{0,\varepsilon} \|_{H^1}+\| u_{1,\varepsilon}-\tilde{u}_{1,\varepsilon} \|_{L^2}+\int_0^T \| f_\varepsilon(s,\cdot) \|_{L^2}\, ds\Big]. \quad (1.25)$$

Let us estimate $\| f_\varepsilon(s,\cdot) \|_{L^2}$. We have

$$\| f_\varepsilon(s,\cdot) \|_{L^2} \lesssim \sum_{j=1}^{d} \| \partial_{x_j}\left[\left(h_{j,\varepsilon}(\cdot)-\tilde{h}_{j,\varepsilon}(\cdot)\right)\partial_{x_j}u_\varepsilon(s,\cdot)\right] \|_{L^2}$$
$$\lesssim \sum_{j=1}^{d} \left[\| \partial_{x_j}h_{j,\varepsilon}-\partial_{x_j}\tilde{h}_{j,\varepsilon} \|_{L^\infty}\| \partial_{x_j}u_\varepsilon \|_{L^2}+\| h_{j,\varepsilon}-\tilde{h}_{j,\varepsilon} \|_{L^\infty}\| \partial_{x_j}^2 u_\varepsilon \|_{L^2}\right].$$

In the last inequality, we used the product rule for derivatives and the fact that $\| \partial_{x_j}\left(h_{j,\varepsilon}-\tilde{h}_{j,\varepsilon}\right)\partial_{x_j}u_\varepsilon \|_{L^2}$ and $\| \left(h_{j,\varepsilon}-\tilde{h}_{j,\varepsilon}\right)\partial_{x_j}^2 u_\varepsilon \|_{L^2}$ can be estimated by $\| \partial_{x_j}h_{j,\varepsilon}-\partial_{x_j}\tilde{h}_{j,\varepsilon} \|_{L^\infty}\| \partial_{x_j}u_\varepsilon \|_{L^2}$ and $\| h_{j,\varepsilon}-\tilde{h}_{j,\varepsilon} \|_{L^\infty}\| \partial_{x_j}^2 u_\varepsilon \|_{L^2}$, respectively. We have by assumption that for all $i=1,\dots,d$, the net $(\tilde{h}_{i,\varepsilon})_\varepsilon$ is moderate. The net $(u_\varepsilon)_\varepsilon$ is also moderate as a very weak solution. Thus, there exists $N\in N$ such that

$$\sum_{j=1}^{d} \| \tilde{h}_{j,\varepsilon} \|_{L^\infty}^{\frac{1}{2}} \lesssim \varepsilon^{-N}, \quad (1.26)$$

$$\sum_{j=1}^{d} \| \partial_{x_j}u_\varepsilon \|_{L^2} \lesssim \varepsilon^{-N} \ and \ \| \Delta u_\varepsilon \|_{L^2} \lesssim \varepsilon^{-N}. \quad (1.27)$$

On the other hand, we have that

$$For\ i=1,\dots,d,\ \| h_{i,\varepsilon}-\tilde{h}_{i,\varepsilon} \|_{W^{1,\infty}} \lesssim C_k\varepsilon^k \ forall\ k>0,$$
$$\| u_{0,\varepsilon}-\tilde{u}_{0,\varepsilon} \|_{H^1} \lesssim C_m\varepsilon^m \ forall\ m>0,$$
and
$$\| u_{1,\varepsilon}-\tilde{u}_{1,\varepsilon} \|_{L^2} \lesssim C_n\varepsilon^n \ forall\ n>0.$$

It follows that
$$\| U_\varepsilon(\cdot,t) \|_{L^2} \lesssim \varepsilon^l, \quad (1.28)$$
for all $l\in N$.

Remark 1.2 *The assumption that $h_i(x)=h(x)$ for all $i=1,\dots,d$, in Theorem 1.2 can be removed if we know that the solution $u(t,x)$ of the problem (*1.12*) is from the class of distributions, that is, $u(t,\cdot)\in E'(R^d)$ for all $t\in[0,T]$.*

Consistency. Now, we want to prove the consistency of the very weak solution with the classical one, when the latter exists, which means that, when the coefficients and the Cauchy data are regular enough, the very weak solution converges to the classical one in an appropriate norm.

Theorem 1.3 (Consistency) *Let $h\in[W^{1,\infty}(R^d)]^d$ be positive. Assume that $u_0\in H^2(R^d)$ and $u_1\in H^1(R^d)$, and let us consider the Cauchy problem*

$$\{u_{tt}(t,x)-\sum_{j=1}^{d}\partial_{x_j}\left(h_j(x)\partial_{x_j}u(t,x)\right)=0,\ \ (t,x)\in[0,T]\times R^d, u(0,x)=u_0(x),\ \ u_t(0,x)=u_1(x),\ \ x\in R^d. \quad (1.29)$$

Let $(u_\varepsilon)_\varepsilon$ be a very weak solution of (1.29). Then, for any regularising families $h_{j,\varepsilon}=h_j*\psi_{1,\varepsilon}$ with $j=1,\dots d$, $u_{0,\varepsilon}=u_0*\psi_{2,\varepsilon}$ and $u_{1,\varepsilon}=u_1*\psi_{3,\varepsilon}$ for any $\psi_k\in C_0^\infty$, $\psi_k\geq 0$, $\int\psi_k=1$, $k=1,2,3$, the net $(u_\varepsilon)_\varepsilon$ converges to the classical solution of the Cauchy problem (1.29) in L^2 as $\varepsilon\to 0$.

Proof. Let u be the classical solution. It solves

$$\{u_{tt}(t,x) - \sum_{j=1}^{d} \partial_{x_j}\left(h_j(x)\partial_{x_j}u(t,x)\right) = 0,\ \ (t,x) \in [0,T] \times R^d, u(0,x) = u_0(x),\ \ u_t(0,x) = u_1(x),\ \ x \in R^d,$$

and let $(u_\varepsilon)_\varepsilon$ be the very weak solution. It solves

$$\{\partial_t^2 u_\varepsilon(t,x) - \sum_{j=1}^{d} \partial_{x_j}\left(h_{j,\varepsilon}(x)\partial_{x_j}u_\varepsilon(t,x)\right) = 0,\ \ (t,x) \in [0,T] \times R^d, u_\varepsilon(0,x) = u_{0,\varepsilon}(x),\ \ \partial_t u_\varepsilon(0,x) = u_{1,\varepsilon}(x),\ \ x \in R^d.$$

Let us denote by $V_\varepsilon(t,x) := u_\varepsilon(t,x) - u(t,x)$. Then V_ε solves the problem

$$\{\partial_t^2 V_\varepsilon(t,x) - \sum_{j=1}^{d} \partial_{x_j}\left(h_{j,\varepsilon}(x)\partial_{x_j}V_\varepsilon(t,x)\right) = \beta_\varepsilon(t,x),\ \ (t,x) \in [0,T] \times R^d, V_\varepsilon(0,x) = (u_{0,\varepsilon} - u_0)(x),\ \ \partial_t V_\varepsilon(0,x) = (u_{1,\varepsilon} - u_1)(x),\ \ x \in R^d,$$

where $$\beta_\varepsilon(t,x) := \sum_{j=1}^{d} \partial_{x_j}\left[(h_{j,\varepsilon}(x) - h_j(x))\partial_{x_j}u(t,x)\right].$$

Once again, using Duhamel's principle and similar arguments as in Theorem 1.3, we arrive at

$$\| V_\varepsilon(\cdot,t) \|_{L^2} \lesssim \left(1 + \sum_{j=1}^{d} \| h_{j,\varepsilon} \|_{L^\infty}^{\frac{1}{2}}\right)\left[\| u_{0,\varepsilon} - u_0 \|_{H^1} + \| u_{1,\varepsilon} - u_1 \|_{L^2} + \int_0^T \| \beta_\varepsilon(s,\cdot) \|_{L^2}\, ds\right], \quad (1.30)$$

where β_ε is estimated by

$$\| \beta_\varepsilon(s,\cdot) \|_{L^2} \lesssim \sum_{j=1}^{d}\left[\| \partial_{x_j}h_{j,\varepsilon} - \partial_{x_j}h_j \|_{L^\infty} \| \partial_{x_j}u \|_{L^2} + \| h_{j,\varepsilon} - h_j \|_{L^\infty} \| \partial_{x_j}^2 u \|_{L^2}\right]. \quad (1.31)$$

Since $\| h_{j,\varepsilon} - h_j \|_{W^{1,\infty}} \to 0$ as $\varepsilon \to 0$ and that u is a classical solution, it follows that the right hand side in the last inequality tends to 0 as $\varepsilon \to 0$. Thus

$$\| \beta_\varepsilon(s,\cdot) \|_{L^2} \to 0 \ \ as\ \varepsilon \to 0. \quad (1.32)$$

From the other hand, for all $j = 1, \ldots, d$ the coefficients $h_{j,\varepsilon}$ are bounded since $h \in [W^{1,\infty}(R^d)]^d$ and we have that

$$\| u_{0,\varepsilon} - u_0 \|_{H^1} \to 0, \quad (1.33)$$

and

$$\| u_{1,\varepsilon} - u_1 \|_{L^2} \to 0, \quad (1.34)$$

as ε tends to 0. It follows that $(u_\varepsilon)_\varepsilon$ converges to u in L^2.

1.3 Finite difference schemes: 1D and 2D models

In this section, we convert 1-dimensional and 2-dimensional tsunami wave equations into explicit and implicit difference schemes and study these schemes for stability and accuracy.

We introduce space-time grids with steps τ, h in the variables t, x respectively, that are

$$\omega_h^\tau = \{(t_{k,}x_i): t_{k,} = k\tau; x_i = ih; (k,i) \in I\},$$

$$\Omega_h^\tau = \{(t_{k,}x_i): t_{k,} = k\tau; x_i = ih; (k,i) \in J\},$$

where

$$I := \{(k,i) \in Z_+^3: 0 < k < M; 0 < i < N;\}$$

$$J \coloneqq \{(k,i) \in Z_+^3 : 0 \le M; 0 < i \le N;\}$$

and

$$X = hN, T = \tau M.$$

One calculates an approximate solution from discrete points in the time-spatial grid $\Omega\tau h$. On this grid we approximate the problem (1.5) using the finite difference method.

Discretisation. Approximate derivatives by central differences $\frac{\partial^2 u}{\partial t^2} \approx \frac{u_i^{k+1} - 2u_i^k + u_i^{k-1}}{\Delta t^2}$

Similarly for the x and y derivatives.

1.3.1 Explicit scheme

To solve the partial differential equation (1.5) with respect to both space and time. We approximate the time and space derivative by central difference, thus we have

$$\frac{u_i^{k+1} - 2u_i^k + u_i^{k-1}}{\tau^2} - H_i^k \frac{u_{i+1}^k - 2u_i^k + u_{i-1}^k}{h^2} - \left(\frac{H_{i+1}^k - H_i^k}{h}\right)\left(\frac{u_{i+1}^k - u_{i-1}^k}{2h}\right) = f_i^k, \qquad (1.35)$$

or, with $C = \tau/h$

$$u_i^{k+1} = 2u_i^k - u_i^{k-1} + C^2 H_i^k \left(u_{i+1}^k - 2u_i^k + u_{i-1}^k\right) + \tau C \left(H_{i+1}^k - H_i^k\right) \frac{C\tau}{2} \left(u_{i+1}^k - u_{i-1}^k\right) + h^2 \tau^2 f_i^k \qquad (1.36)$$

Schematic representation of the scheme (1.39) is shown on Fig. 1.2.

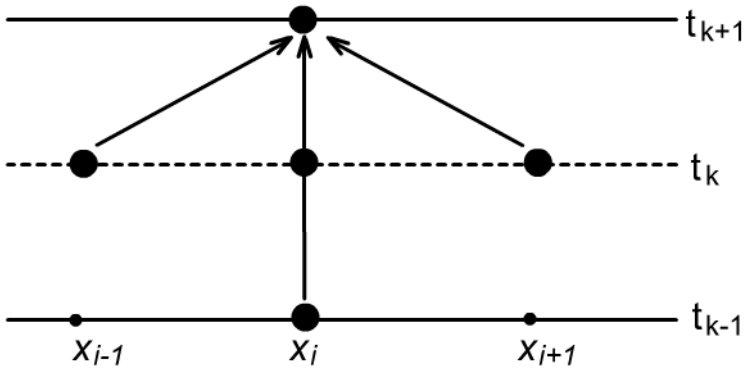


Figure 1.2 – Schematic visualization of the numerical scheme (1.36) for (1.5)

for $(k,i) \in \omega_h^\tau$, where $H_i^k := h(k\tau, ih), f_i^k := f(k\tau, ih)$ with initial conditions

$$u_i^0 = \varphi_i, u_i^1 - u_i^{-1} = 2\tau\psi_i$$

For(i)$\in \mathbf{0}, \mathbf{N}$ and with boundary conditions

$$u_0^k = 0, u_N^k = 0$$

In order to compute u_i^{k-1} we need to implement the second initial condition

$$u_t(0, x_i) = \psi(x_i) = \frac{u_i^1 - u_i^{-1}}{2\Delta t} + O(\tau^2)$$

With $\psi_i := \psi(x_i)$ one can rewrite the last expression as

$$u_i^{-1} = u_i^1 - 2\tau\psi_i + O(\tau^2)$$

and the second time row can be calculated as

$$u_i^1 = u_i^0 + \tau\psi_i + \frac{C^2}{2}H_i^0\left(u_{i+1}^0 - 2u_i^0 + u_{i-1}^0\right) + \tau C\left(H_{i+1}^0 - H_i^0\right)\frac{C\tau}{2}(u_{i+1}^0 - u_{i-1}^0) + \frac{h^2\tau^2 f_i^0}{2}$$

To solve the equation (1.39), we use the simple iterative method.

von Neumann Stability Analysis

Here, we consider stability analyses for numerical scheme (1.36).

$$u_j^{k+1} = 2u_j^k - u_j^{k-1} + C^2H\left(u_{j+1}^k - 2u_j^k + u_{j-1}^k\right) + \frac{C\tau}{2}\frac{\partial H}{\partial x}(u_{j+1}^k - u_{j-1}^k)$$

In order to investigate the stability of the scheme we start with the usual ansatz

$$u_j^k = \xi^k e^{i\beta j\Delta x},$$

$$\xi^{k+1}e^{i\beta j\Delta x} = 2\xi^k e^{i\beta j\Delta x} - \xi^{k-1}e^{i\beta j\Delta x} + C^2H\xi^k\left(e^{i\beta(j+1)\Delta x} - 2e^{i\beta j\Delta x} + e^{i\beta(j-1)\Delta x}\right) + \frac{C\tau}{2}\frac{\partial H}{\partial x}(e^{i\beta(j+1)\Delta x} - e^{i\beta(j-1)\Delta x})$$

Then we divide both sides on $\xi^k e^{i\beta j\Delta x}$

$$\xi = 2 + \xi^{k-1} + C^2H\left(e^{i\beta\Delta x} - 2 + e^{-i\beta\Delta x}\right) + \frac{C\tau}{2}\frac{\partial H}{\partial x}(e^{i\beta\Delta x} - e^{-i\beta\Delta x})$$

$$\xi = 2 + \xi^{k-1} + C^2H(2\cos(\beta\Delta x) - 2) + \frac{C\tau}{2}\frac{\partial H}{\partial x}(2isin(\beta\Delta x))$$

$$\xi^2 - \left(2 + 2C^2H\left(-2sin^2\left(\frac{\beta\Delta x}{2}\right)\right) + C\tau\frac{\partial H}{\partial x}(isin(\beta\Delta x))\right)\xi + 1 = 0,$$

which leads to the following expression for the amplification factor $\xi(\beta)$

After that the last expression becomes just a quadratic equation for ξ, namely

$$\xi^2 - \left(2 + 4C^2H\left(sin^2\left(\frac{\beta\Delta x}{2}\right)\right) + C\tau\frac{\partial H}{\partial x}(isin(\beta\Delta x))\right)\xi + 1 = 0,$$

$$a = c = 1, b = 2 + 4C^2H\left(sin^2\left(\frac{\beta\Delta x}{2}\right)\right) + C\tau\frac{\partial H}{\partial x}(isin(\beta\Delta x))$$

$$\xi_1 \cdot \xi_2 = 1$$

Possibilities $\xi_1 > 1, \xi_2 < 1, \xi_1 = \xi_2 = 1$

For stability $|\xi| \leq 1$ so $\xi_1 = \xi_2 = 1$ is true.

Solutions of the equation for ξ (β) read

$$\xi_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}; \xi_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Roots are equal to 1 only when roots are complex

$$b^2 - 4ac < 0$$

$$\left[2 + 4C^2H\left(sin^2\left(\frac{\beta\Delta x}{2}\right)\right) + C\tau\frac{\partial H}{\partial x}\left(isin(\beta\Delta x)\right)\right]^2 - 4 < 0$$

$$4\left[1 + 2C^2H\left(sin^2\left(\frac{\beta\Delta x}{2}\right)\right) + \frac{C\tau}{2}\frac{\partial H}{\partial x}\left(isin(\beta\Delta x)\right)\right]^2 < 4$$

$$\mathrm{C}(2\mathrm{C}H\left(sin^2\left(\tfrac{\beta\Delta x}{2}\right)\right) + \tfrac{\tau}{2}\tfrac{\partial H}{\partial x}\left(isin(\beta\Delta x)\right)) < -2$$

$$C < \frac{-2 - \frac{\tau}{2}\frac{\partial H}{\partial x}\left(isin(\beta\Delta x)\right)}{Hsin^2\left(\frac{\beta\Delta x}{2}\right)}$$

This is true always if C<1

That is, the scheme is conditional stable. The stability condition reads

$$C < \frac{-2 - \frac{\tau}{2}\frac{\partial H}{\partial x}\left(isin(\beta\Delta x)\right)}{Hsin^2\left(\frac{\beta\Delta x}{2}\right)}$$

1.3.2 Implicit scheme

For simplicity, consider the Crank-Nicolson scheme for the problem (1.5) which is the average of the central differences about the point (i) and $(i, k+1)$. First we transform the partial differential equation (1.5) into the implicit finite-difference equation then we get

$$\frac{u_i^{k+1} - 2u_i^k + u_i^{k-1}}{\tau^2} - \frac{H_i^k}{2h^2}\left(u_{i+1}^{k+1} - 2u_i^{k+1} + u_{i-1}^{k+1} + u_{i+1}^{k-1} - 2u_i^{k-1} + u_{i-1}^{k-1}\right) - \left(\frac{H_{i+1}^k - H_i^k}{h}\right)\left(\frac{u_{i+1}^k - u_{i-1}^k}{2h}\right) = f_i^k \qquad (1.37)$$

Schematic diagram of the numerical scheme (1.37) is shown on Fig. (1.3).

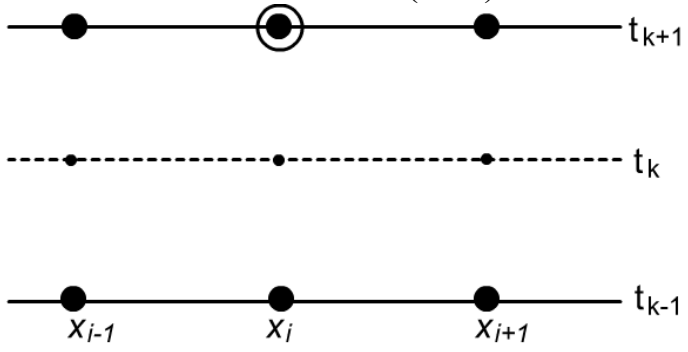


Figure 1.3 – Schematic visualization of the implicit numerical scheme (1.37) for (1.5)

The equation (1.40) can be reduced to the most general form:

$a_i u_{i+1}^{k+1} - b_i u_i^{k+1} + c_i u_{i-1}^{k+1} = f_i,$

where

$$a_i = \tau^2 H_{i,}\ b_i = 2h^2 + 2\tau^2 H_{i,}\ c_i = \tau^2 H_{i,}$$

$$f_{\ i} = -4h^2 u_i^k + 2h^2 u_i^{k-1} - \tau^2 H_i \left(u_{i+1}^{k-1} - 2u_i^{k-1} + u_{i-1}^{k-1}\right) + 2\tau^2 (H_{i+1} - H_i)\left(u_{i+1}^k - u_i^k\right) - 2h^2\tau^2 f_i.$$

The system has a tridiagonal structure so we solve it using by Thomas method and the conditions for the correctness and stability of the Thomas method are like this:

$$|b_i| > |a_i| + |c_i| \quad \forall i = \underline{1, N-1}$$

Here, we consider stability analyses for numerical scheme (1.37).

In order to investigate the stability of the scheme we start with the usual ansatz

$$u_j^k = \xi^k e^{i\beta j\Delta x},$$

$$\xi^{k+1} e^{i\beta j\Delta x} = 2\xi^k e^{i\beta j\Delta x} - \xi^{k-1} e^{i\beta j\Delta x} + C^2 H\left(\xi^{k+1} e^{i\beta(j+1)\Delta x} - 2\xi^{k+1} e^{i\beta j\Delta x} + \xi^{k+1} e^{i\beta(j-1)\Delta x} + \xi^{k-1} e^{i\beta(j+1)\Delta x} - 2\xi^{k-1} e^{i\beta j\Delta x} + \xi^{k-1} e^{i\beta(j-1)\Delta x}\right) + \frac{\tau}{2} \mathrm{C} \frac{\partial H}{\partial x} (\xi^k e^{i\beta(j+1)\Delta x} - \xi^k e^{i\beta(j-1)\Delta x})$$

Then we divide both sides on $\xi^k e^{i\beta j\Delta x}$

$$\xi = 2 - \xi^{-1} + C^2 H \left(\xi\left(e^{i\beta\Delta x} - 2 + e^{-i\beta\Delta x}\right) + \xi^{-1}\left(e^{i\beta\Delta x} - 2 + e^{-i\beta\Delta x}\right)\right) + \frac{\tau}{2} \mathrm{C} \frac{\partial H}{\partial x} (e^{i\beta\Delta x} - e^{-i\beta\Delta x})$$

which leads to the following expression for the amplification factor $\xi(\beta)$

After that the last expression becomes just a quadratic equation for ξ, namely

$$\xi^2 \left(1 - 2HC sin^2\left(\frac{\beta\Delta x}{2}\right)\right) - \xi(2 + \tau C \frac{\partial H}{\partial x} isin(\beta\Delta x)) + (1\text{-}2\mathrm{H}\tau C sin\frac{\beta\Delta x}{2}) = 0$$

After solving the quadratic equation and some reductions we get

$$|\xi|^2 = 1$$

That is, the implicit scheme (1.37) is *absolutely stable*.

for $(k, i) \in \omega_h^\tau$, where $H_i^k := h(k\tau, ih), f_i^k := f(k\tau, ih)$ with initial conditions

$$u_i^0 = \varphi_i, u_i^1 - u_i^0 = \tau\varphi_i,$$

For(i)∈ **0**, **N** and with boundary conditions

$$u_0^k = 0, u_N^k = 0,$$

for $(i,k) \in 0,N \times 0,M$, respectively. It is well-known, that the implicit scheme (1.37) is unconditionally stable and it has accuracy order $O(\tau + |h|^2)$, see, for example [16].

1.3.3 Numerical Accuracy Analysis of 1D equation

The numerical solution requires a certain level of numerical accuracy. We check the accuracy of we used the finite-difference scheme using absolute error and norm error L^2 which is defined as

$$err_{abs} = ||u - u_{num}||,$$
$$L^2 = \sqrt{\sum_{i=1}^{n}(u - u_{num})^2 \Delta x \Delta y},$$

where u denotes the exact value and u_{num} denotes the approximation

When H=1 the original problem (1.6) has an exact solution with initial conditions

$$u(0,x) = Asin\frac{\pi x}{l}$$

$$u(t,x) = Asin\frac{\pi x}{l} cos\frac{\pi ct}{l}$$

Table 1.1 – Maximum norm and L^2norm errors

Δx	Δt	Explicit scheme		Implicit scheme	
		max error	$L^2 - norm$	max error	$L^2 - norm$
0.1	0.2	9.77658e+2	4.58484 e+2	3.23973e-1	1.89892e-1
0,01	$0{,}01^2$	3.93016e-2	2.29719-2	3.93029e-3	2.29719 e-3
0,001	$0{,}001^2$	5.43811e-2	1.00549 e-2	5.43828e-3	1.00549 e-3
0,0001	$0{,}0001^2$	3.44479e-3	1.73587 e-4	3.44154e-4	1.73589 e-4

Table 1.1 displays the convergence rate of displacement solution under grid refinement. The convergence rates in maximum norm at the final time shows forth order convergence

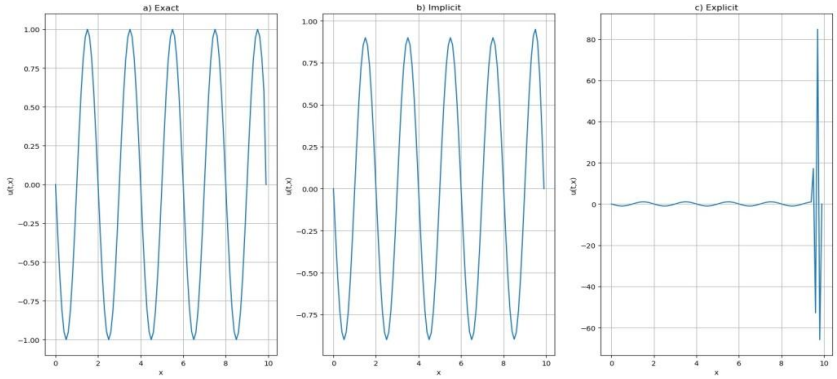


Figure 1.4 – Comparison of exact and numerical solutions when $t = 1, \Delta x = 0.1, \Delta t = 0.2$, a) exact solution b) solution using the implicit scheme c) solution using the explicit scheme

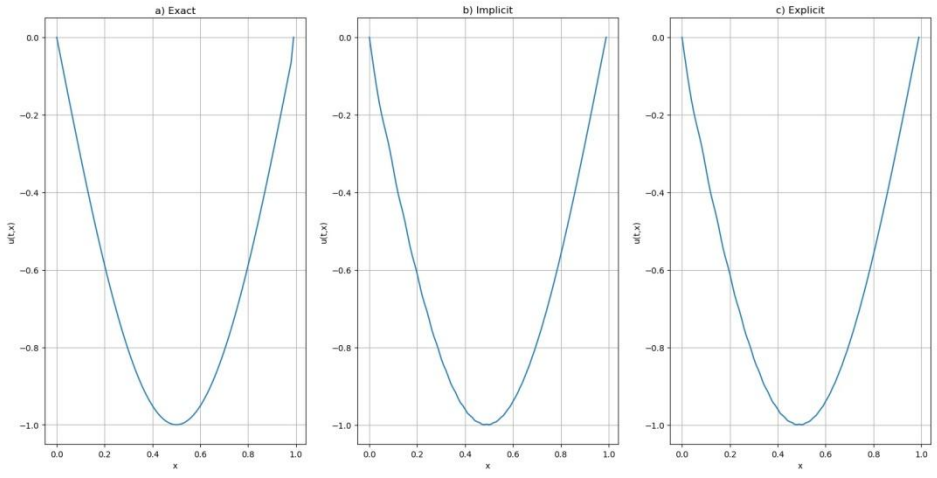


Figure 1.5 – Comparison of exact and numerical solutions when $t = 1, \Delta x = 0.01, \Delta t = 0.0001$, a) exact solution b) solution using the implicit scheme c) solution using the explicit scheme

From Table 1 and Figures 1.4 and 1.5 we see that the explicit difference scheme is not stable at large time steps and the implicit difference scheme is stable at large time steps therefore, in all our calculations, we use an indistinct difference scheme.

The 2D case

The two-dimensional tsunami model

$$\begin{cases} u_{tt}(t,x,y) - [\partial_x\big(H(t,x,y)\partial_x u(t,x,y)\big) + \partial_y\left(H(t,x,y)\partial_y u(t,x,y)\right)] + \\ \partial_{tt}H(t,x,y) = f(t,x,y), (\mathrm{t},\mathrm{x},\mathrm{y}) \in (0,\mathrm{T}) \times \Omega \\ \mathrm{u}(0,\mathrm{x},\mathrm{y}) = \varphi(\mathrm{x},\mathrm{y}), u_t(0,x,y) = \psi(x,y), (\mathrm{x},\mathrm{y}) \in \Omega \\ \mathrm{u}(\mathrm{t},\mathrm{x},\mathrm{y})|\,\partial\Omega = \mathrm{g}(\mathrm{t},\mathrm{x},\mathrm{y}), \mathrm{t} \in [0,\mathrm{T}]. \end{cases} \quad (1.38)$$

We consider this equation on the rectangular space domain. We introduce space-time grids with steps τ, h_1, h_2 in the variables t, x, y respectively that are

$$\omega^{\tau}_{h_1 h_2} = \left\{\left(t_{k,}x_i, y_j\right) : t_{k,} = kt; x_i = ih_1; y_j = jh_2; (k,i,j) \in I\right\},$$

$$\Omega^{\tau}_{h_1 h_2} = \left\{\left(t_{k,}x_i, y_j\right) : t_{k,} = kt; x_i = ih; y_j = jh; (k,i,j) \in J\right\},$$

where

$$I \coloneqq \{(k,i,j) \in Z^3_+ : 0 < k < M; 0 < i < N_1; 0 < j < N_2\},$$

$$J \coloneqq \{(k,i,j) \in Z^3_+ : 0 < k \leq M; 0 < i \leq N_1; 0 < j \leq N_2\},$$

and

$$X = h_1 N_1, Y = h_2 N_2, T = \tau M.$$

When H=1 (1.38) equation has analytical solution

We adopt the unit square $(x, y) \in [0; 1] \times [0; 1]$ as the spatial solution domain with 100 elements per each side and 100 interior points, c = 1, with initial condition

$$u(x, y, 0) = sin(2\pi x)sin(2\pi y), \qquad \frac{\partial u(x, y, 0)}{\partial t} = 0,$$

and $u(0, y, t) = u(1, y, t) = u(x, 0, t) = u(y, 1, t) = 0$ on the boundaries. The analytical solution of equation (1.38) is as flows: $u(x, y, t) = cos(\sqrt{2}\pi t)sin(2\pi x)sin(2\pi y)$.

Explicit scheme

To solve the partial differential equation (1.5) with respect to both space and time. We approximate the time and space derivative by central difference, for simplicity, we take $N_1 = N_2$ $h_1 = h_2 = h$ and thus we have

$$\frac{u_{i,j}^{k+1} - 2u_{i,j}^{k} + u_{i,j}^{k-1}}{\tau^2} - H_{i,j}^{k}\left(\frac{u_{i+1,j}^{k} - 2u_{i,j}^{k} + u_{i-1,j}^{k}}{h^2} + \frac{u_{i,j+1}^{k} - 2u_{i,j}^{k} + u_{i,j-1}^{k}}{h^2}\right) + \\ + \left(\frac{H_{i+1,j}^{k} - H_{i,j}^{k}}{h}\right)\left(\frac{u_{i+1,j}^{k} - u_{i,j}^{k}}{h}\right) - \left(\frac{H_{i,j+1}^{k} - H_{i,j}^{k}}{h}\right)\left(\frac{u_{i,j+1}^{k} - u_{i,j}^{k}}{h}\right) = f_{i,j}^{k}, \qquad (1.39)$$

for $(k, i, j) \in \omega_h^{\tau}$,where $H_{i,j}^{k} := h(ih), f_{i,j}^{k} := f(k\tau, ih, jh)$ with initial conditions

$$u_{i,j}^{0} = \varphi_{i,j}, \qquad u_{i,j}^{1} - u_{i,j}^{0} = \tau\varphi_{i,j},$$

For (i, j) ∈ $\mathbf{0}, \boldsymbol{N}$ and with boundary conditions

$$u_0^k = 0, u_N^k = 0.$$

Implicit scheme

We consider the Crank-Nicolson scheme for the problem (1.38) which is the average of the central differences about the point (i, j) and $(i, j, k + 1)$.

$$\frac{u_{i,j}^{k+1} - 2u_{i,j}^{k} + u_{i,j}^{k-1}}{\tau^2} - \frac{H_{i,j}^{k}}{4h^2}\Big((u_{i+1,j}^{k+1} - 2u_{i,j}^{k+1} + u_{i-1,j}^{k+1} + u_{i+1,j}^{k-1} - 2u_{i,j}^{k-1} + u_{i-1,j}^{k-1}) + \\ \left(u_{i+1,j}^{k+1} - 2u_{i,j}^{k+1} + u_{i-1,j}^{k+1} + u_{i+1,j}^{k-1} - 2u_{i,j}^{k-1} + u_{i-1,j}^{k-1}\right)\Big) + \left(\frac{H_{i+1,j}^{k} - H_{i,j}^{k}}{h}\right)\left(\frac{u_{i+1,j}^{k} - u_{i,j}^{k}}{h}\right) - \\ \left(\frac{H_{i,j+1}^{k} - H_{i,j}^{k}}{h}\right)\left(\frac{u_{i,j+1}^{k} - u_{i,j}^{k}}{h}\right) = f_{i,j}^{k} \qquad (1.40)$$

It is well-known, that the implicit scheme (1.40) is unconditionally stable and it has accuracy order $O(\tau + |h|^2)$.

Numerical Accuracy Analysis of 2D equation.

Graphic comparisons of the exact solution with the numerical and errors are shown in figure 1.6.

Table 1.2 – Maximum norm and L^2norm errors

		Explicit scheme		Implicit scheme	
Δx	Δt	max error	$L^2 - norm$	max error	$L^2 - norm$
0.1	0.2	1.3492e+3	7.2582e+2	1.77618e-1	0.47569e-1

0,01	$0{,}01^2$	4.35651e-3	2.18914e-3	2.45655e-3	1.48964e-3
0,001	$0{,}001^2$	3.52562e-3	6.27457e-4	2.12562e-3	4.38543e-4
0,0001	$0{,}0001^2$	2.87473e-5	1.24878e-6	1.87473e-3	1.04570e-4

Table 1.2 displays the convergence rate of displacement solution under grid refinement. The convergence rates in maximum norm at the final time shows forth order convergence.

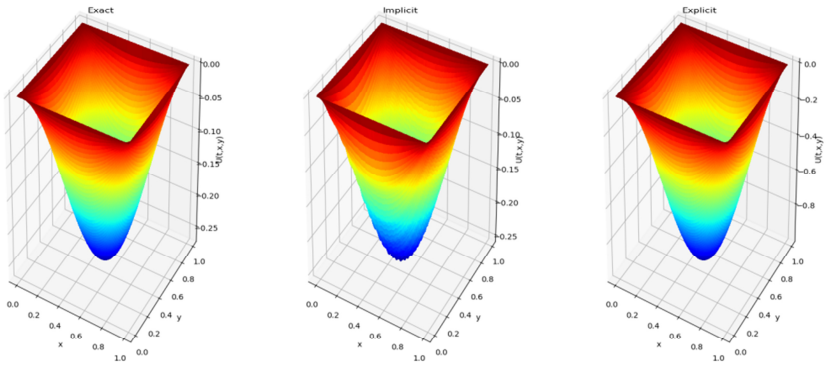


Figure 1.6 – Comparison of exact and numerical solutions when $t = 1, \Delta x = 0.01, \Delta t = 0.0001$, from left to right exact solution, solution using the implicit scheme, solution using the explicit scheme

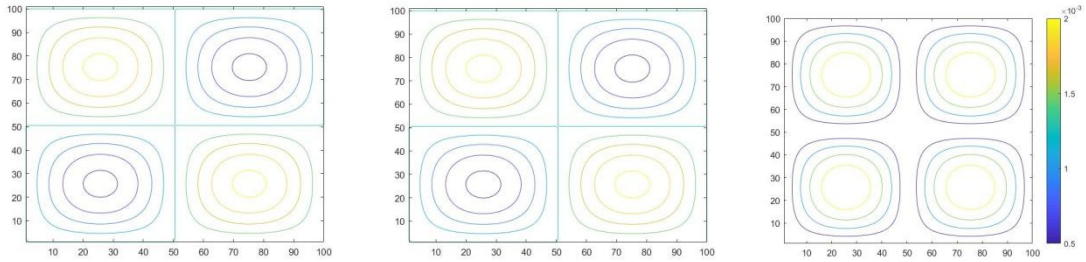


Figure 1.7 – Solution for time t = 1, Δx = Δy = 0,01, Δt = 0,0001 from left to right exact, numerical, error

Table 1.2 and Figures 1.6 and 1.7 shows the implicit difference scheme is stable at large time steps.

1.4 Caspian sea tsunami study

History has shown that tsunamis occur not only in large seas but also in small seas. As described in the introductory section, there have been many earthquakes in our Caspian Sea, and each year there are earthquakes of different magnitudes, resulting in tsunami waves, which have been studied and predicted by scientists from Russia, Iran and Azerbaijan. Here, scientists from these countries were engaged only in modeling their part of the Caspian Sea. No one has studied the Kazakh side and predicted how

high it will reach the coast in the event of a tsunami. In this section, we will model the Aktau port and on the Caspian Sea using the model we studied, and make various predictions.

To model a possible tsunami in the port of Aktau, we need a submarine bathymetric map of the Caspian Sea, which we obtain from the National Centers for Environmental Information portal[17] and a bathymetric map of the area we are going to model is shown in figure 1.8.

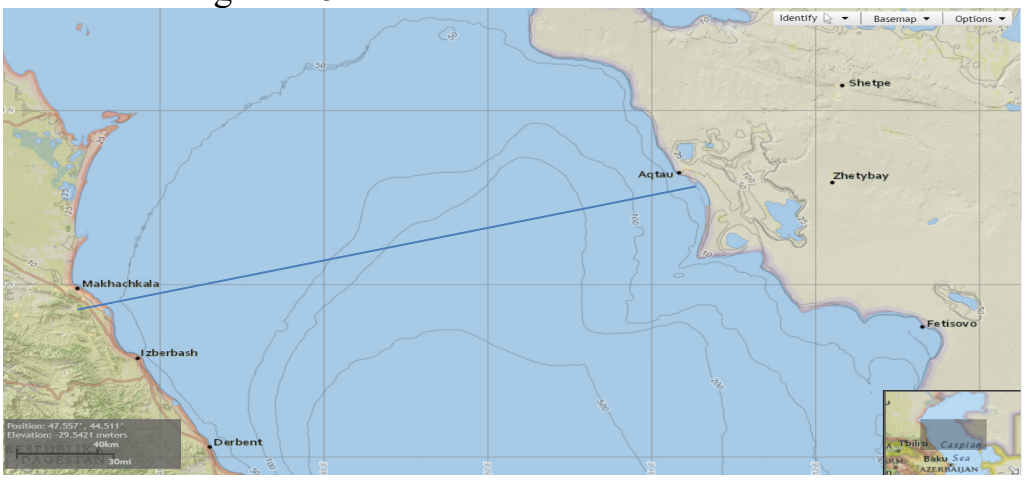


Figure1.8 – Bathymetric map of Caspian Sea

The city of Aktau is very close to the sea, as shown in Figure 1.9; 3 kilometers from the sea there are many infrastructure, kindergartens, schools, residential areas, shopping centers, so tsunami risk modeling is very important in this area.

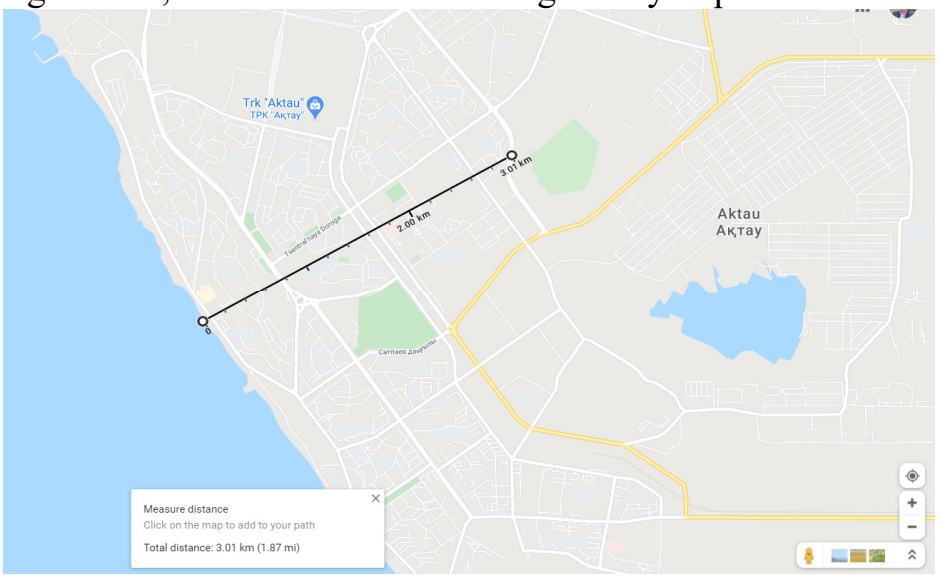


Figure 1.9 – Aktau coast

As a mathematical model, we get the following model described above.

$$u_{tt}(t,x,y) - [\partial_x\big(h(x,y)\partial_x u(x,y)\big) + \partial_y\left(h(x,y)\partial_y u(t,x,y)\right)] = f(t,x,y), (t,x,y) \in (0,T)\times\Omega$$

Here H is the sea bottom topology which we get from the bathymetric map, as a initial condition, we get a function that describes the first wave that occurs after an earthquake. As a boundary condition, we use Dirichlet boundary condition.

The main focus here is to predict how many meters the tsunami wave will reach the shore.

Here we simulate the Caspian Sea tsunami between Aktau and Makhachkala, as shown in the map below fig 1.10, the distance between Aktau and Makhachkala is 300 km. Suppose that the initial wave from an underwater earthquake occurred between Aktau and Makhachkala, then we can predict, relative to the height of the first wave, how high the wave will reach Aktau and Makhachkala.

Figure 1.10 – Distance between Aktau and Makhachkala and epicenter of the initial wave

The numerical method used for 1D and 2D cases is very similar. Space and time are divided into a grid. The size of the basin area was determined as 300 km by 300 km. The distance between each grid point Δx and Δy was set equal to 0.5 km. The time step size was set $\Delta t = 1/200$ s. We used a finite difference method to approximate the 2D tsunami equation. The spatial derivatives were approximated using second-order centered differences. Time derivatives were fitted using second-order centered differences according to the Crank- Nicolson method.

As shown in Figure 1.8 from the Bathymetric Map of the Caspian Sea, we can approximately determine the function of the basin profile as follow

$$H(x) = \begin{cases} 0{,}4x, 0 \le x < 25 \\ 10 + 1{,}8(x-25), 25 \le x < 75 \\ 100 + 2(x-75), 75 \le x < 125 \\ 200, 125 \le x < 150 \\ 200 - 1{,}25(x-150), 150 \le x < 230 \\ 100 - 5/3(x-230), 230 \le x < 260 \\ 50 - 4/3(x-260), 260 \le x < 290 \\ 10 - (x-290), 290 \le x < 300 \end{cases}$$

Such that the depth would range from 0 m to 200 m. We made the depth of water to be positive at all x since very high numerical error occurs if we let depth profile be

negative-this would correspond to above sea level. The initial condition was given as a Gaussian U.

$$U(0,x) = 20exp(-(x-150)^2/10)$$

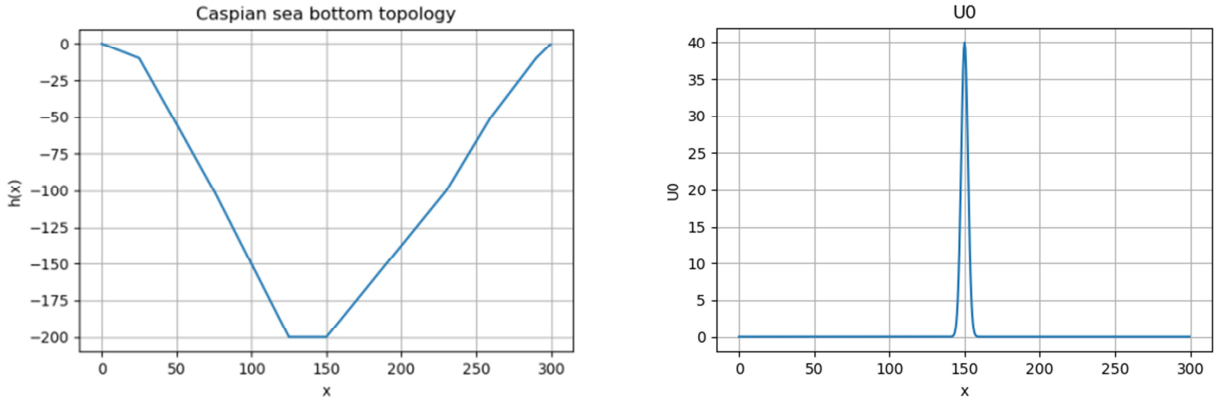


Figure 1.11 – Plot of –h, the depth of profile and initial condition

The results of this simulation are shown in Figure 1.12. As the tsunami approached the coast, the wave slows down and the altitude increased.

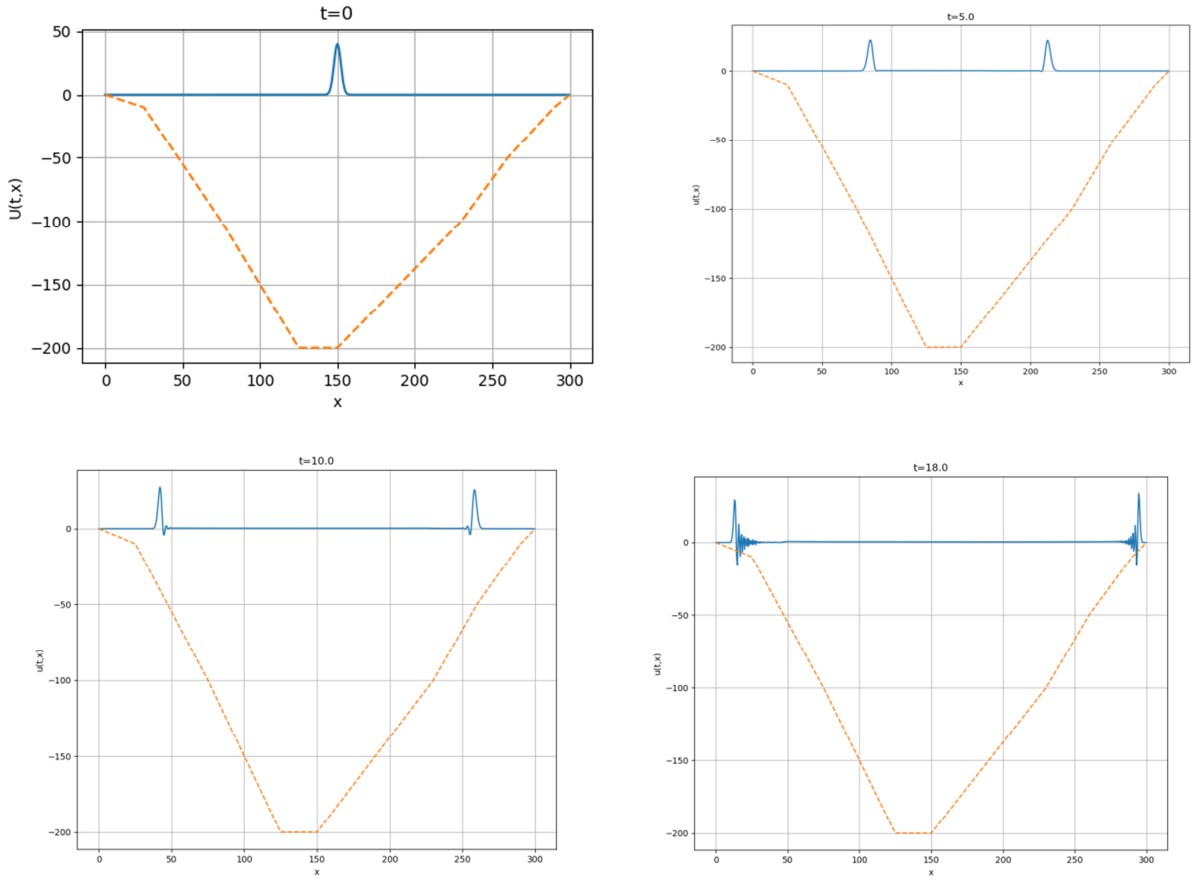


Figure 1.12 – 1D simulation results with depth profile at various time step

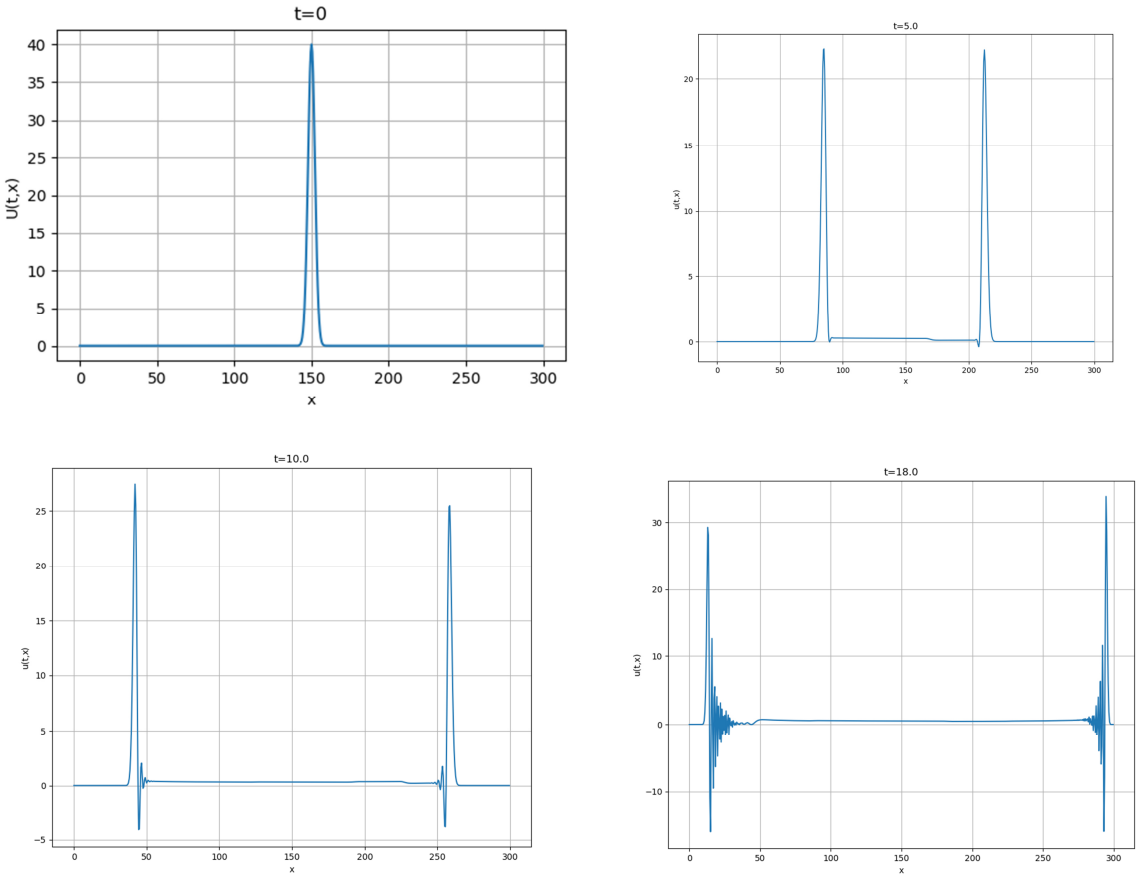


Figure 1.13 – 1D simulation results without depth profile at various time step

The 2D results given as below

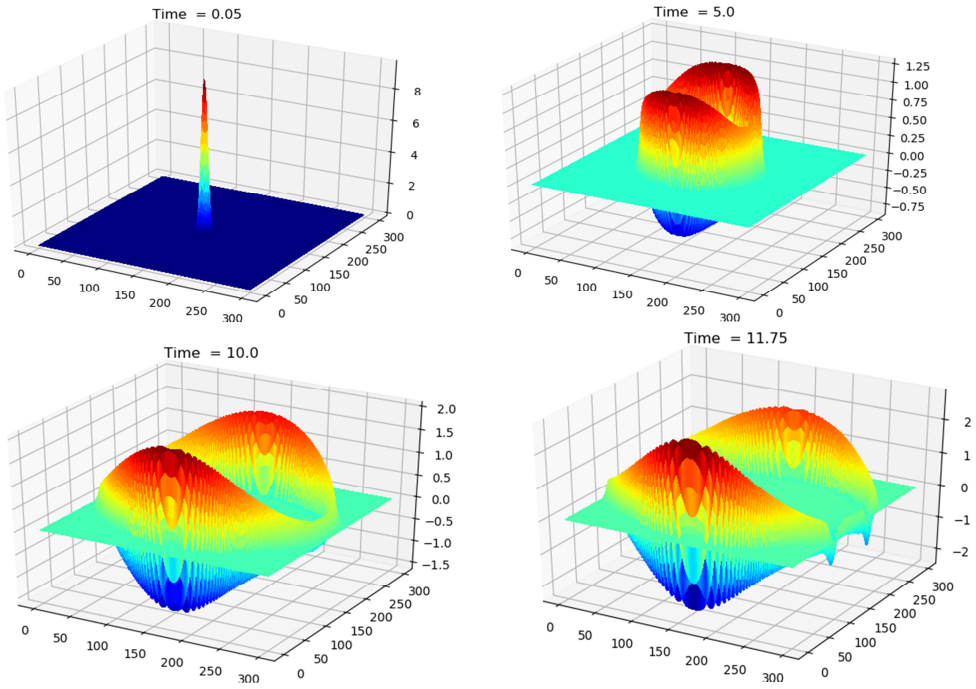


Figure 1.14 – 2D simulation results without depth profile at various time steps

Different predictions related to the height of the initial wave

To get the real value of the wave height, we have to convert the dimensionless form to dimensional using the following formulations:

$$\underline{x} = xL; \underline{y} = yL; \underline{t} = tT; \underline{u} = u\underline{E}.$$

Tsunami Speed Formula

$$\vartheta = \sqrt{gh}.$$

Where g acceleration of gravity, h is water depth.

If we propose the initial wave occur in 150 km from Aktau city the tsunami wave will reach to the coastal area in 56 minute.

Table 1.3 – Calculation result

Initial wave height	The wave height in the sea shore
2 m	2.94 m
3 m	4.4 m
4 m	5.89 m
8 m	11.75m

Considering that the shores of the Caspian Sea in Aktau are 1-3 m above the water level. From the forecasts above, it can be seen that a strong earthquake in the middle of the sea will cause great damage to the city of Aktau.

1.5 Numerical results

In this section, we numerically solve the 1D and 2D equations of tsunami and acoustic waves and visualize the results. Too solve numerically we use finite difference schemes that are shown in the previous section.

1.5.1 Numerical results of tsunami wave equation

1D tsunami wave equation. To solve the partial differential equation (1.5) we use (1.6) and (1.7) finite difference schemes we consider two particular cases of the coefficient h(x). Here we allow them to be distributional, in particular, to have δlike singularities. As it was theoretically outlined in [6-7] we start to analyse our problem by regularising distributions h(x) by a parameter ϵ, that is, we set

$$h_\varepsilon(x) := (h * \varphi_\varepsilon)(x)$$

as the convolution with the mollifier

$$\varphi_\varepsilon(x) = \frac{1}{\varepsilon}\varphi(x/\varepsilon)$$

With

$$h_1(x) = \begin{cases} 100;\ 0 \le x < 70 \\ 100\ 9(x - 70);\ 70 \le x < 80 \\ 10;\ 80 \le x < 100 \end{cases}$$

$$h_2(x) = \begin{cases} 100;\ 0 \le x < 75 \\ 10;\ 75 \le x < 100 \end{cases}$$

For all simulations we take $\Delta t = 0{,}05$, $\Delta x = 0{,}5$

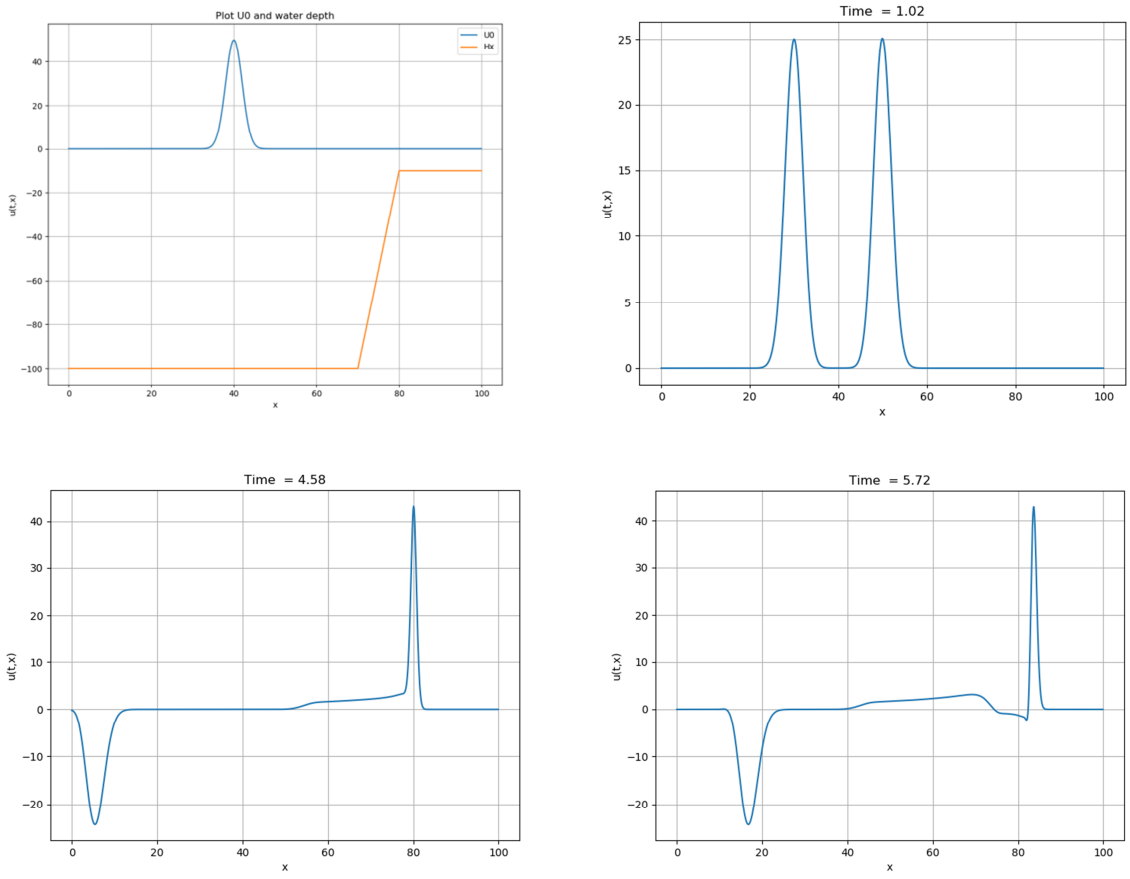
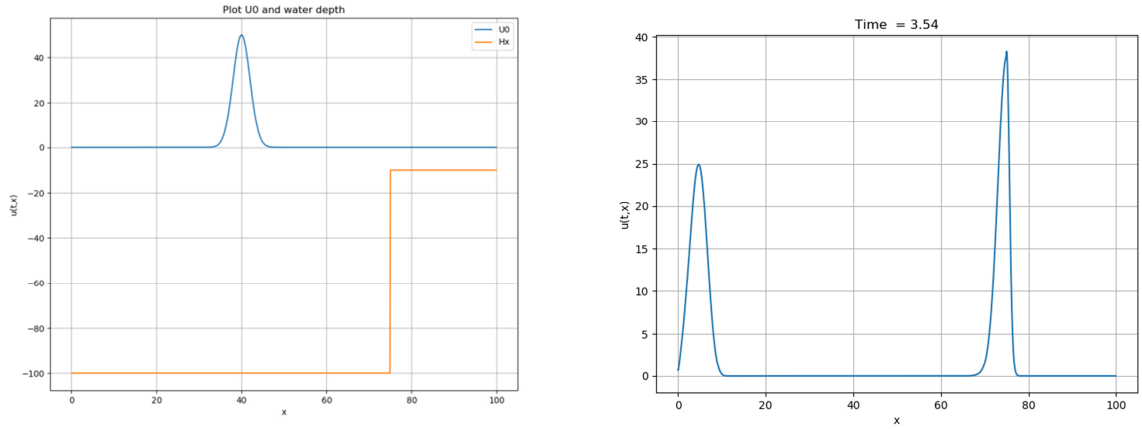


Figure 1.15 – Case 1 displacement of wave at different times



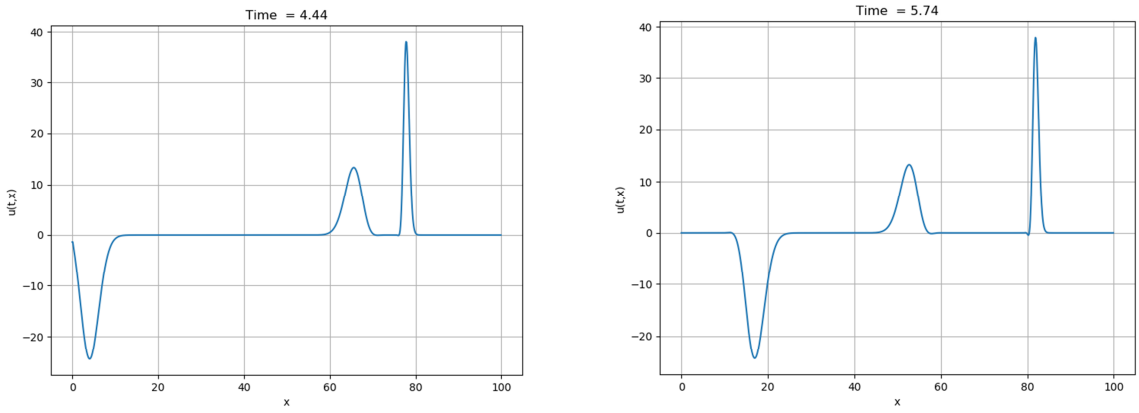


Figure 1.16 – Case 2 displacement of wave at different times

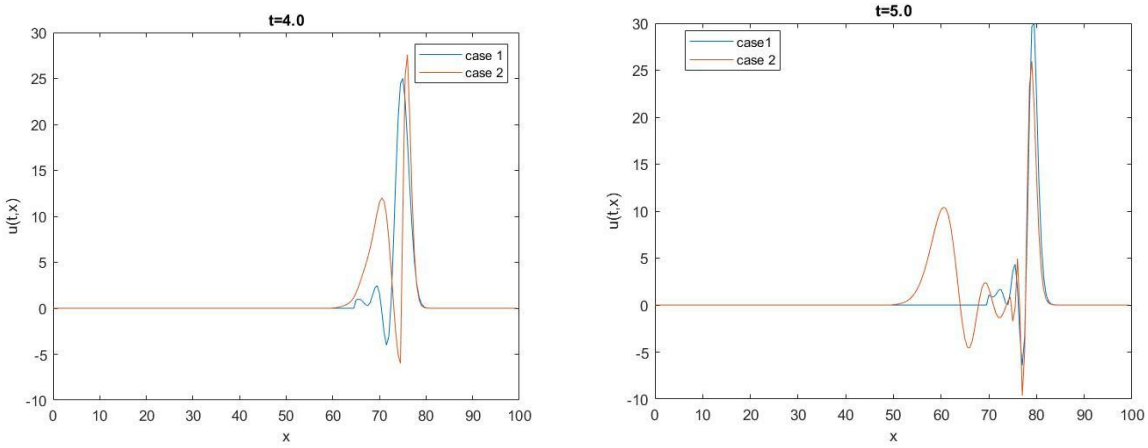


Figure 1.17 – Comparison of both cases for time t=4.0 and t=5.0

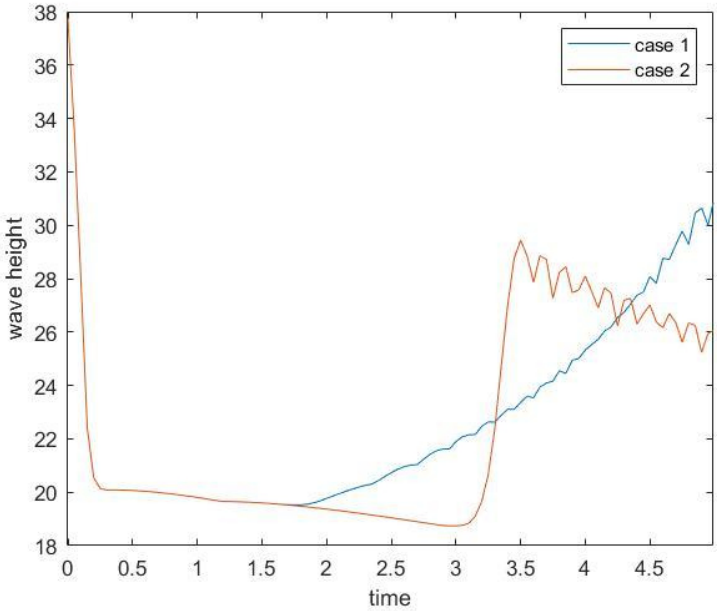


Figure 1.18 – Comparison of wavelengths in both cases

Figures 1.15-1.18 show the results of simulations obtained at different times as a H(x) function we use the first and second cases. in the second case, the singularity is greater than in the first case, as we can see in the figure, a small part of the wave goes backward, and the wave height decreases slightly, and in the first case the singularity is small and the wave does not go back, but the wave height also increases.

2D tsunami wave equation. As the 1D case for H we get same cases, we do the same simulation for 2D case

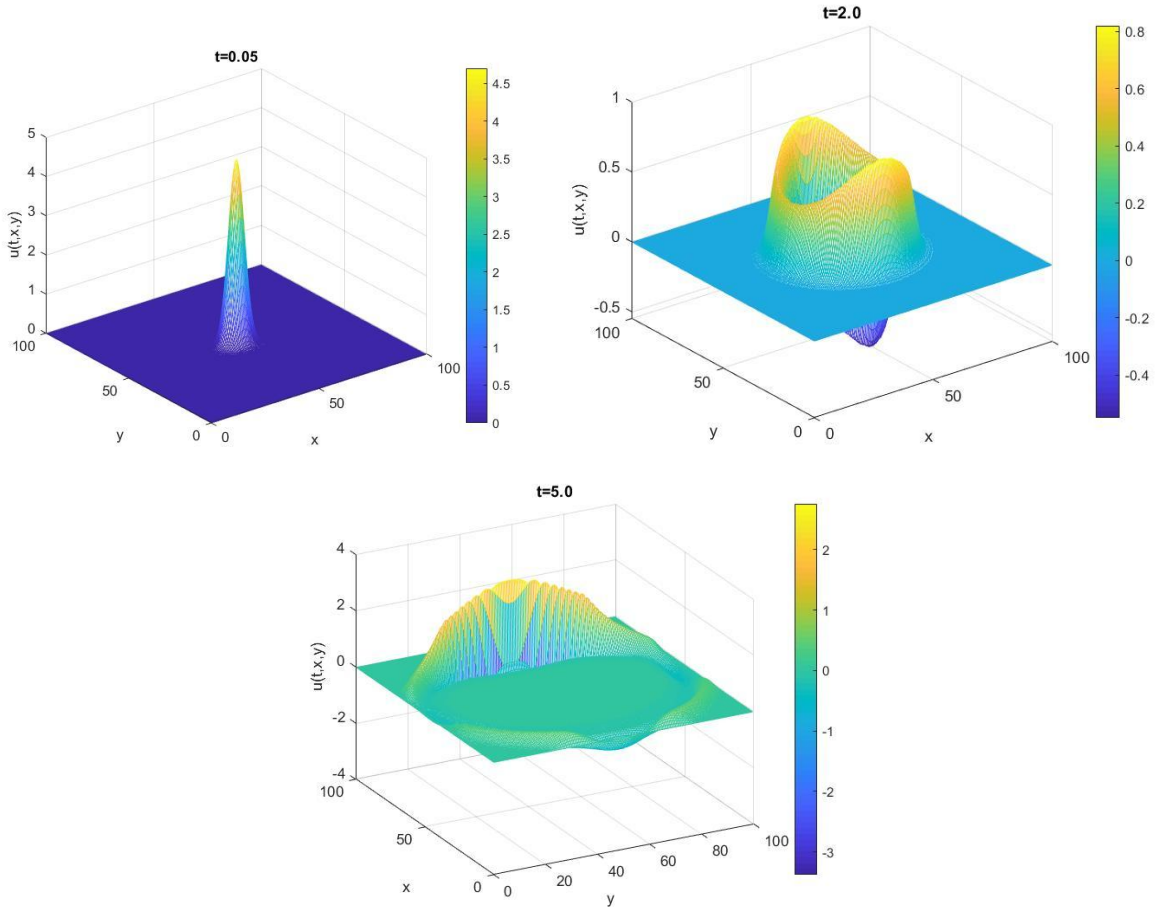


Figure 1.19 – Case 1 displacement of waves at different times

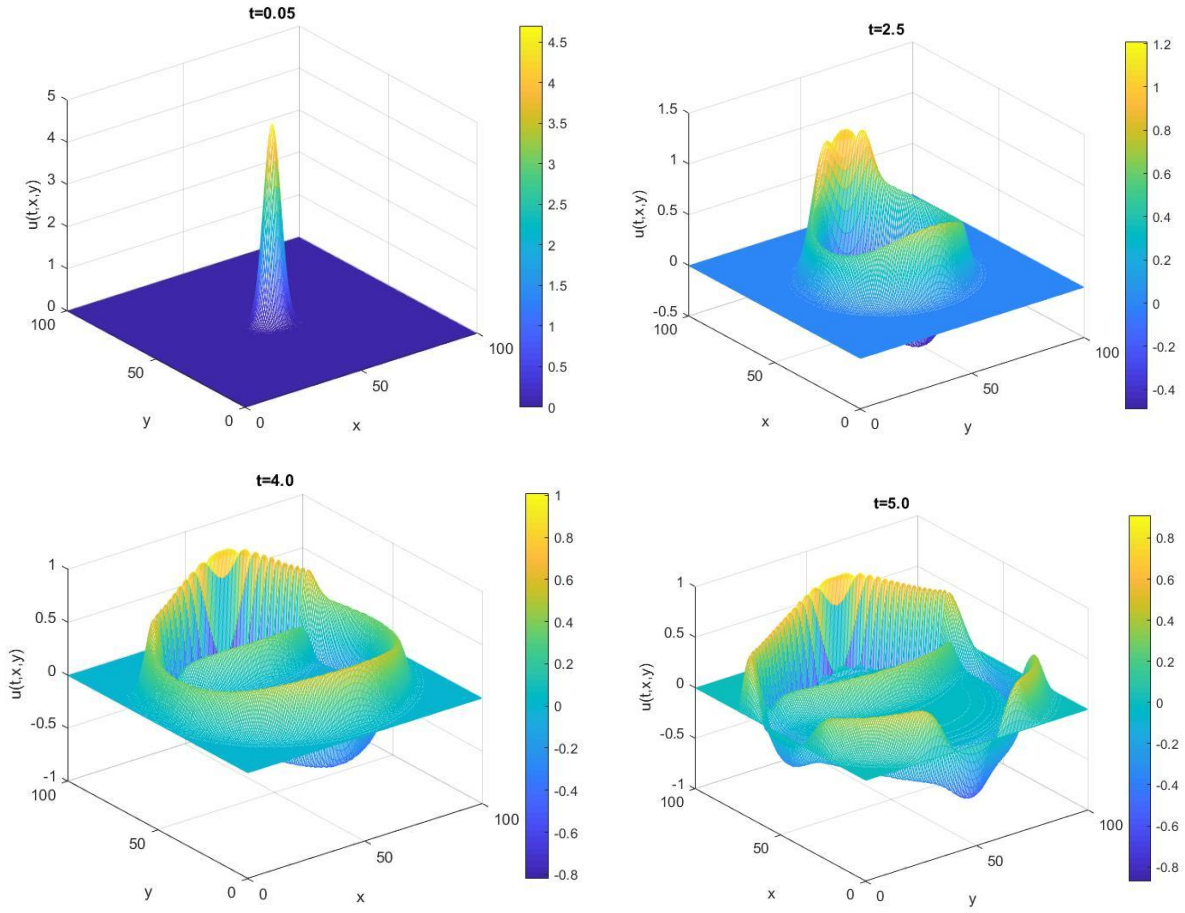


Figure 1.20 – Case 1 displacement of wave at different times

In the two-dimensional model, we also consider the same two cases for the function H (x, y) that we used in the one-dimensional case, and from these figures, we can clearly see the small part of the wave return back in the case of a higher singularity.

1.5.2 Numerical simulations of the 1d wave equation with a distributional coefficient and source term

In this section, we illustrate numerical experiments for the one-dimensional wave equation with δ-like terms. Our research is connecting the theory with the numerical realisations. By using results on very weak solutions introduced by Ruzhansky with his co-authors, we investigate a corresponding regularized problem. In contrast to our expectations, the experiments show that the solution of the regularized problem has a "good" behavior[83].

Here, we study the Cauchy-Dirichlet problem for the 1D-Wave Equation

$$\begin{cases} \partial_t^2 u(t,x) - a(t)\partial_x^2 u(t,x) + q(t)u(t,x) = 0, \qquad (t,x) \in [0,T] \times [0,10], \\ u(t,0) = 0, \ \ t \in [0,T], \\ u(t,10) = 0, \ \ t \in [0,T], \\ u(0,x) = u_0(x), \ \ x \in [0,10], \\ \partial_t u(0,x) = u_1(x), \ \ x \in [0,10]. \end{cases} \tag{1.41}$$

In this work we consider several particular cases of the coefficient $a(t)$ and the mass term $q(t)$. Here we allow them to be distributional, in particular, to have δ–like singularities.

As it was theoretically outlined in [6] and [18], we start to analyse our problem by regularising distributions $a(t)$ and $q(t)$ by a parameter ε, that is, we set

$$a_\varepsilon(t) := (a * \varphi_\varepsilon)(t), \quad q_\varepsilon(t) := (q * \varphi_\varepsilon)(t),$$

as the convolution with the mollifier

$$\varphi_\varepsilon(t) = \frac{1}{\varepsilon}\varphi(t/\varepsilon),$$

with

$$\varphi(t) = \{\frac{1}{c} e^{1/(t^2+1)}, |t| \le 1, 0, |t| > 1,$$

where $c \simeq 0.443994$ to get $\int_{-1}^{1} \varphi(t)dt = 1$. Then, instead of (1.41) we consider the regularised problem

$$\begin{cases} \partial_{tt}^2 u_\varepsilon(t,x) - a_\varepsilon(t)\partial_{xx}^2 u_\varepsilon(t,x) + q_\varepsilon(t)u_\varepsilon(t,x) = 0, \ (t,x) \in [0,T] \times [0,10], \\ u_\varepsilon(t,0) = 0, \ t \in [0,T], \\ u_\varepsilon(t,10) = 0, \ t \in [0,T], \\ u_\varepsilon(0,x) = u_0(x), \ x \in [0,10], \\ \partial_t u_\varepsilon(0,x) = 0, \ x \in [0,10]. \end{cases} \tag{1.42}$$

Here, we put $u_1(x) \equiv 0$ and

$$u_0(x) = \begin{cases} e^{1/((x-4.1)^2 - 0.1)}, \ |x - 4.1| < 0.1, \\ 0, \ |x - 4.1| \ge 0.1. \end{cases}$$

Note that $supp\ u_0 \subset [4,4.2]$.

For a and q we consider the following combinations of the possible cases, with δ denoting the standard Dirac's delta-distribution:

- $a = 1, q = 0$;
- $a = 1 + 5\delta(t-3), q = 0$;
- $a = 1 + 5\delta(t-3), q = 1$;
- $a = 1 + 5\delta(t-3), q = 10\delta(t-7)$;
- $a = 1 + 5\delta(t-3), q = 1 + 10\delta(t-7)$;
- $a = 1 + 5\delta(t-3), q = 1 + 5\delta'(t-1.5)$;

In Figure 1.21, we compare solutions of the problem (1.42) in different cases. In the upper-left plot, we compare the behaviours of the solution corresponding to the cases

(A1) and (A2) coloured by blue and red, respectively, at $t = 3.2$ for $\varepsilon = 0.1$. In the upper-right plot, we compare the behaviours of the solution corresponding to the cases (A1) and (A3) coloured by blue and red, respectively, at $t = 3.2$ for $\varepsilon = 0.1$. In the bottom plot, we compare the behaviour of the solutions of the problem (1.42) corresponding to the cases (A1) and (A6) coloured by blue and red, respectively, at $t = 3.2$ for $\varepsilon = 0.1$. Here when the mass term q is positive, we see that the wave level is lower than when it is absent. But when $q = 1 + 5\delta'(t - 1.5)$, it can be interpreted as a quickly changeable mass (not only volume but also its sign), and we get less stable waves, as it is shown in the plot.

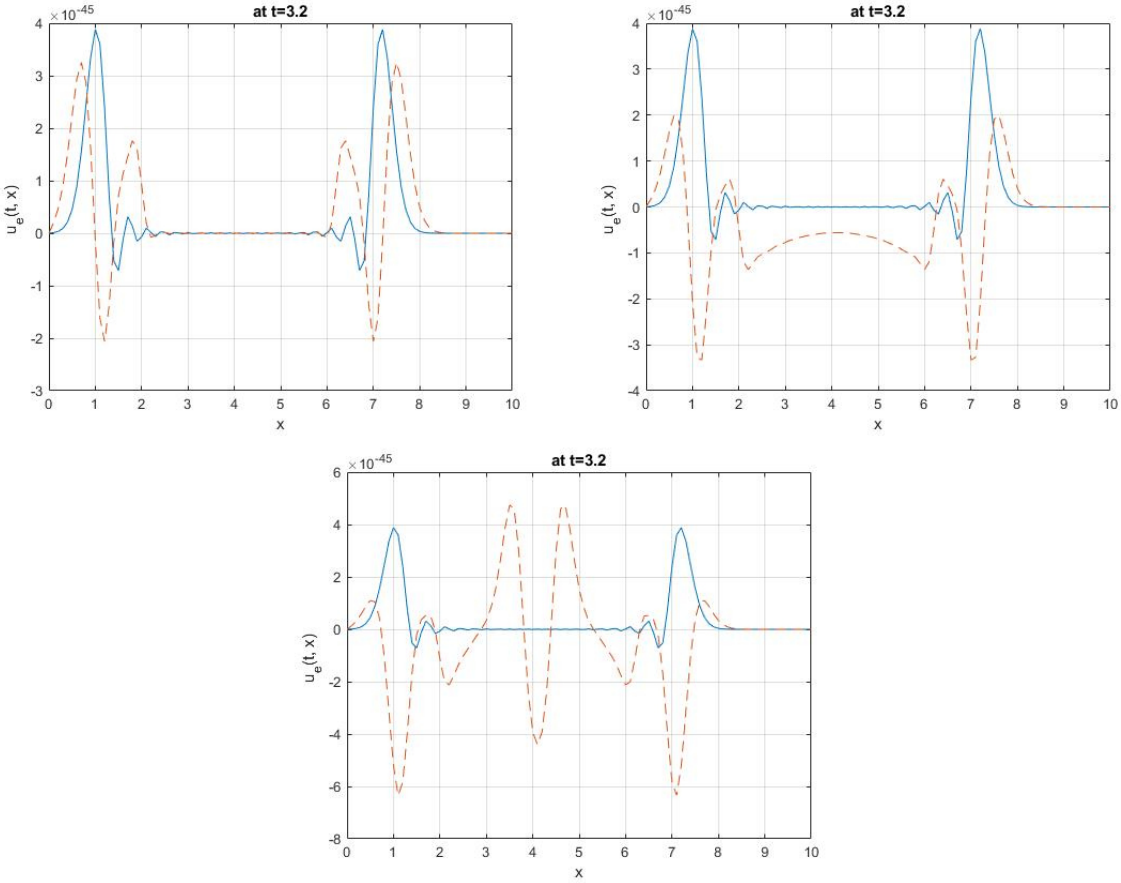


Figure 1.21 – In all plots, from right to left we see the solution of the problem (1.42) in the case (A1) coloured by blue at $t = 3.2$ and $\varepsilon = 0.1$. Red lines: in the upper-left plot, we see the solution of the problem (1.45) in the case (A2); in the upper-right plot, we see the solution of the problem (1.45) in the case (A3); in the bottom plot, we see the solution of the problem (1.42) in the case (A6) at $t = 3.2$, for $\varepsilon = 0.1$

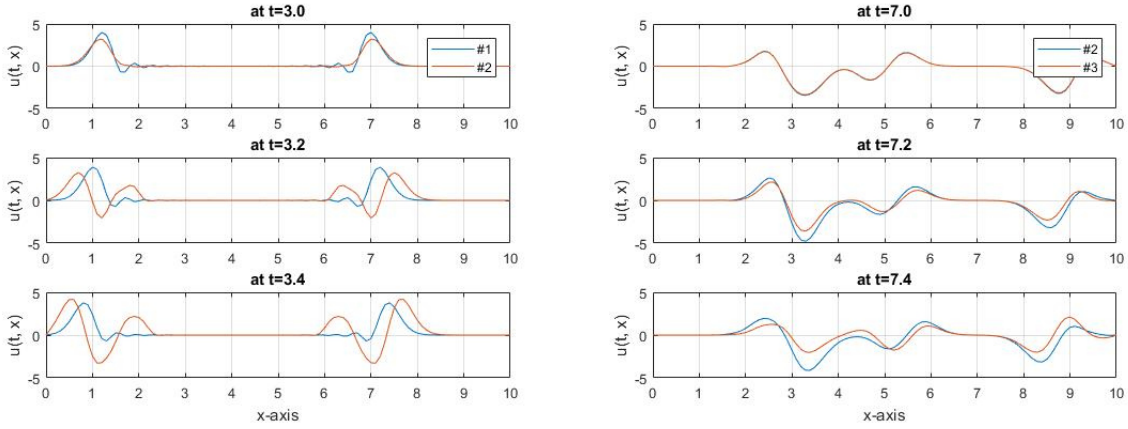


Figure 1.22 – In the left plot, from up to bottom we see the solution of the problem (1.42) coloured by blue in the case (A1) and by red in the case (A2) at $t = 3,3.2,3.4$. In the right plot, from up to bottom we can see the solution of the problem (1.42) coloured by blue in the case (A2) and by red in the case (B4) at $t = 7,7.2,7.4$, for $\varepsilon = 0.1$

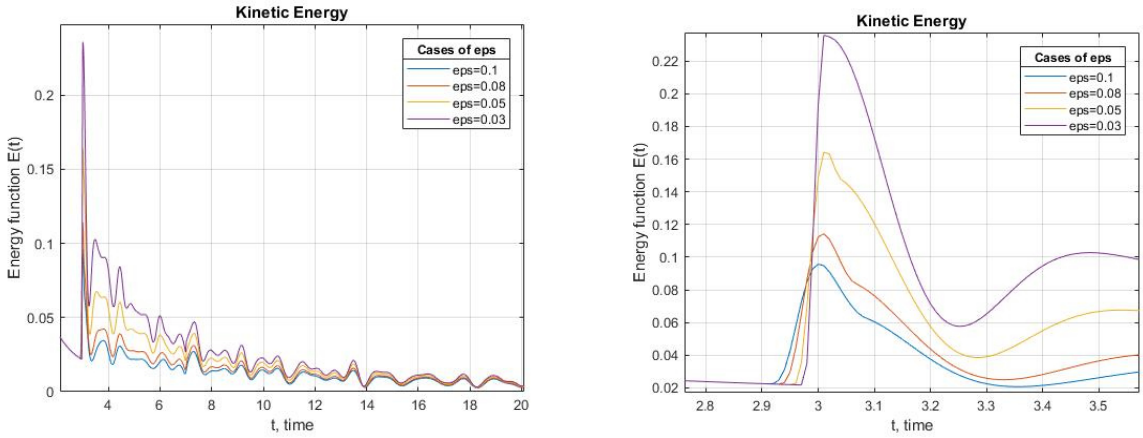


Figure 1.23 – In the both plots, we compare the kinetic energy of the system corresponding to the problem (1.42) in the case (B5) for different values of $\varepsilon = 0.03,0.05,0.08,0.1$. In the right plot we focus near $t = 3$ and can see an influence to the energy of the singular propagation speed: the propagation speed effects comparatively more than the mass term

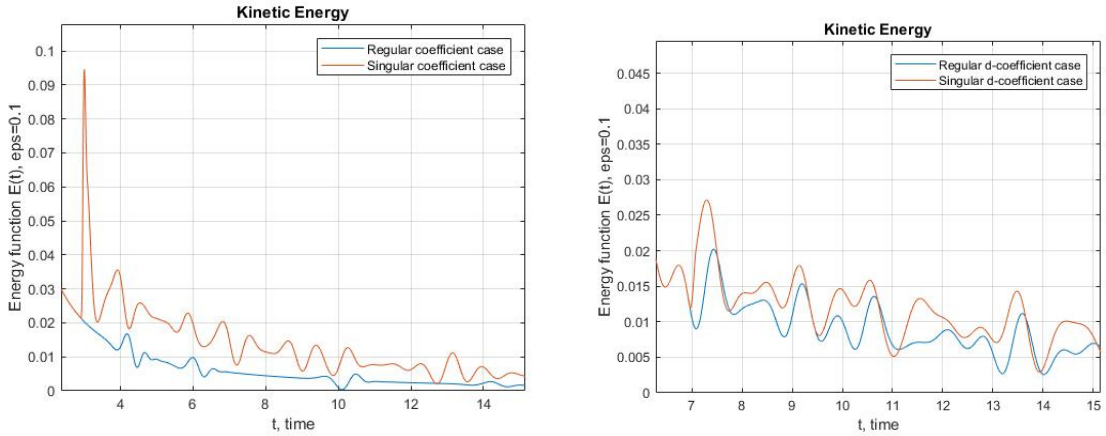


Figure 1.24 – In the left plot, we compare the kinetic energy of the system corresponding to the problem (1.42) in the cases (A1), coloured by blue, and (A2), which is coloured by red, for $\varepsilon = 0.1$. In the right plot, we compare the kinetic energy of the system corresponding to the problem (1.42) in the cases (A3), coloured by blue, and (B5), which is coloured by red, for $\varepsilon = 0.1$

In Figure 1.22 in the left plot, from up to bottom we see the solution of the problem (1.42) coloured by blue in the case (A1) and by red in the case (A2) at $t = 3,3.2,3.4$. In the right plot, from up to bottom we can see the solution of the problem (1.42) coloured by blue in the case (A2) and by red in the case (B4) at $t = 7,7.2,7.4$, for $\varepsilon = 0.1$.

In Figure 1.23, we compare the "kinetic energy" $E(t) = \int_0^{10} |\partial_t u_\varepsilon(x,t)|^2 dx$ of the system corresponding to the problem (1.42) in the case (B5) for different values of $\varepsilon = 0.03,0.05,0.08,0.1$. In Figure 4 in the left plot, we compare the kinetic energy of the system corresponding to the problem (1.42) in the cases (A1), coloured by blue, and (A2), which is coloured by red, for $\varepsilon = 0.1$. In the right plot, we compare the kinetic energy of the system corresponding to the problem (1.42) in the cases (A3), coloured by blue, and (B5), which is coloured by red, for $\varepsilon = 0.1$. The analysis shows that the kinetic energy of the singular problems is higher than the problems without δ–like terms. However, in the both cases the kinetic energy decays in time. In the left plot, we analyse the behaviour of the kinetic energy, in particular, how it depends on the parameter ε. Even if the energy function shows impulses at the shocked moments, in general, it decays in time.

All numerical computations are made in C++ by using the sweep method. In above numerical simulations, we used the Matlab R2017b. For all simulations we take $\Delta t = 0.01$, $\Delta x = 0.1$.

The analysis carried out in this section showed that the numerical methods work well in the situations where a rigorous mathematical formulation of the problem is difficult within the classical theory of distributions. The concept of very weak solutions eliminates this difficulty, giving the well-posedness results for equations with δ–like coefficients. Numerical experiments showed that a notion of very weak solutions introduced in [4] is very well adapted for numerical simulations. Moreover, by the recently constructed theory of very weak solutions we can talk about uniqueness of the numerical solutions to differential equations with δ–like coefficients in a suitable sense.

1.5.3 Numerical results of acoustic wave equation - homogeneous case

In this section, we simulate the propagation of a two-dimensional acoustic wave equation in a homogeneous and heterogeneous medium.

In Homogeneous case, the acoustic wave equation is given by

$$\frac{\partial^2 P(t,x,y)}{\partial t^2} = c^2 \left(\frac{\partial^2 P(t,x,y)}{\partial x^2} + \frac{\partial^2 P(t,x,y)}{\partial y^2} \right) + S(t,x,y)$$

Where P is pressure, S is source term, c is wave velocity.

The difference schemes of this equation are the same as the tsunami wave equation, the only difference in the coefficient so in this section we just present numerical simulations.

Numerical results as follow:

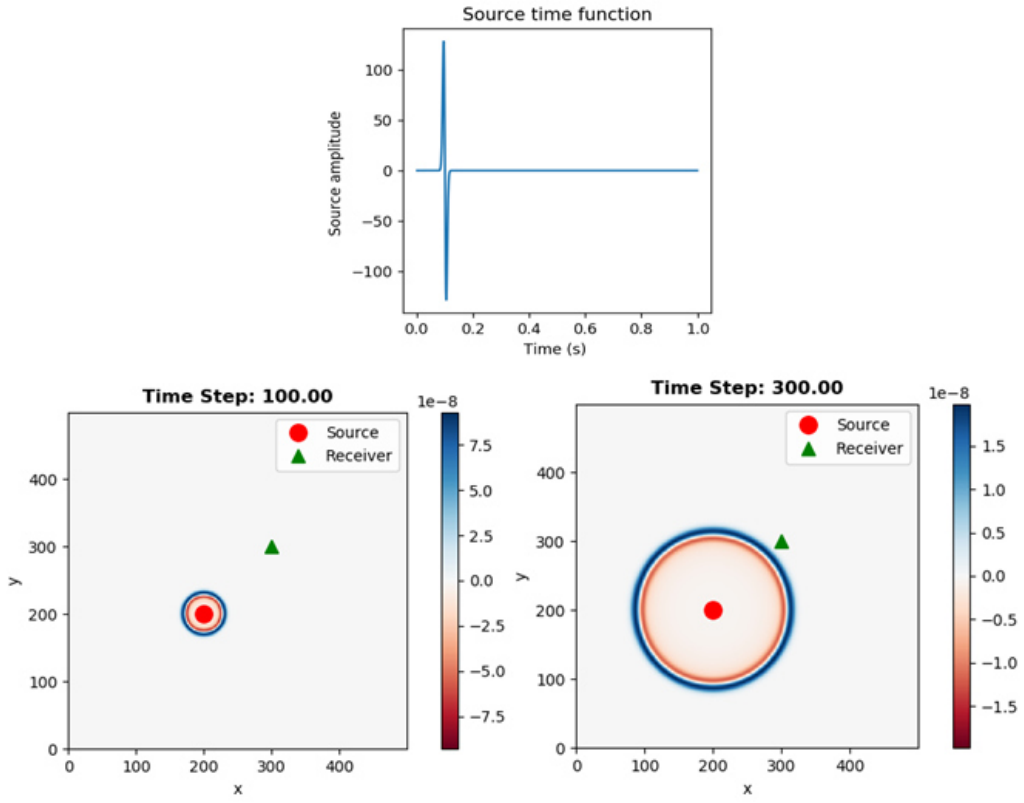


Figure1.25 – Numerical result for homogeneous case

In heterogeneous case, the acoustic wave equation is given by

$$\frac{\partial^2 P(t,x,y)}{\partial t^2} = c^2(x,y)\left(\frac{\partial^2 P(t,x,y)}{\partial x^2} + \frac{\partial^2 P(t,x,y)}{\partial y^2}\right) + S(t,x,y)$$

An heterogeneous medium, the wave velocity varies depending on the structure of the medium. Here we consider the propagation of waves the following medium is shown in figure 1.26, where the wave velocities differ in the white and blue regions, for example, in our case, in the blue region, the wave speed is 580 m/s, and in the white region, 464 m /s.

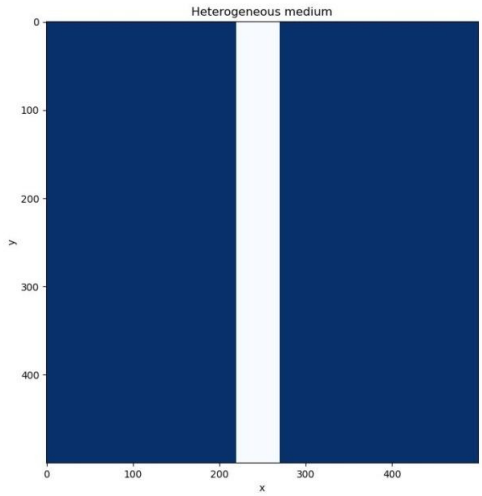


Figure 1.26 – Simple heterogeneous medium

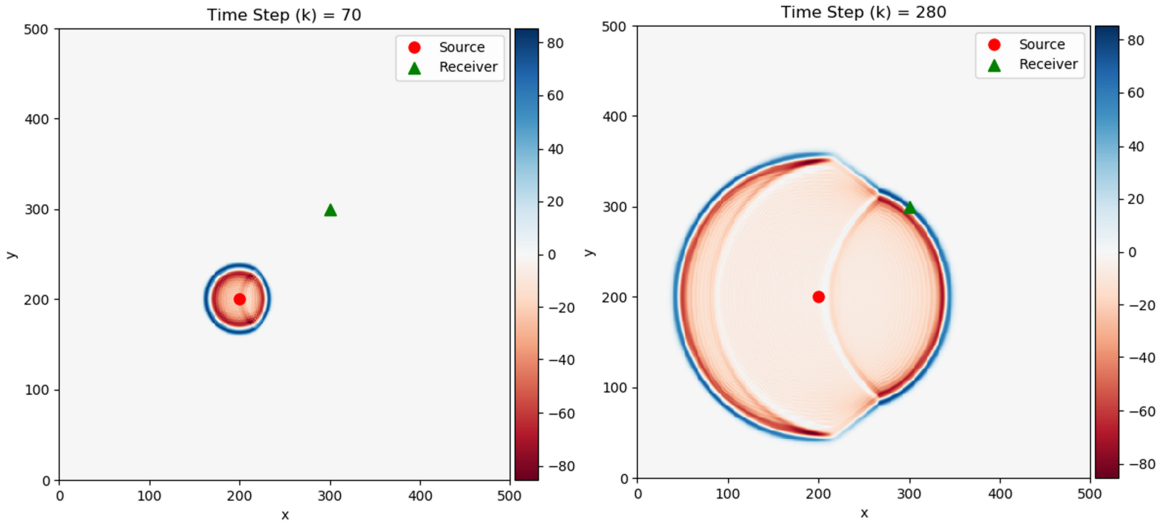


Figure 1.27 – Numerical result for heterogeneous case

In Figure 1.27, we see the propagation of an acoustic wave in a non-homogeneous medium, from which we find that when sound travels to another medium, its velocity changes and occurs the phenomenon of reflection.

1.6 Conclusion

In this chapter, we have described mathematical models of tsunami and acoustic wave equations, given the proof of existence, uniqueness and convergence of the very weak solution to the tsunami wave equation. And constructed difference schemes for one-dimensional and two-dimensional cases and studied these schemes for stability and accuracy.

Modeled Caspian tsunami by the tsunami equation and made different predictions related to the height of the initial wave. At the end of this chapter, we made numerical simulations 1D and 2D tsunami and acoustic wave equations for 2 different cases.

2 REVIEW OF PARALLEL NUMERICAL IMPLEMENTATION OF HYPERBOLIC TYPE EQUATIONS

In the second chapter, we present a parallel numerical implementation of hyperbolic wave equations. Here we propose 3 parallel algorithms, and the third algorithm has published in the scientific journal [82].

In the first algorithm, we consider the MPI implementation of the two-dimensional wave equation with a singular coefficient, and the second algorithm presents the CUDA implementation of the two-dimensional tsunami equation.

At the end of this chapter, we present a hybrid implementation of the acoustic wave problem, and then compare the results of different implementations. In a hybrid implementation, we jointly use Open MP, CUDA and MPI technologies to solve one problem and present the calculation results.

2.1 The architecture of parallel computing environments

Currently, there are different ways to classify the architecture of parallel computing systems. The first common classification in the literature was proposed by M. Flynn in the late 60s of the last century. Its classification is based on two different stream concepts: data and commands. Here it is proposed to classify the architecture into four classes according to the number of flows.

1. SISD (Single Instruction Single Data) - it is a single-instruction machine architecture that operates with a single thread, that is, a single instruction that executes a single flow. The classic von Neumann machine belongs to this architecture.

SIMD (Single Instruction, Multiple Data) is a computer computation principle that allows parallelism at the data level. SIMD computers are composed of one command processor (control module) called a controller and several data processing modules called processing elements. The control module receives, analyzes and executes commands. If the command contains data, the controller sends the command to all processing elements, and this command is executed on some or all of the processing elements.

3. MISD (Multiple Instruction, Simple Data) is a parallel computing architecture in which two or more functional modules perform different operations on the same data. Many researchers refer to this class of conveyor computers.

4. MIMD (Multiple Instruction, Multiple Data) is a computer architecture in which several independent processors work as part of a larger system. Processing is divided into several streams, each of which has its own processor setup, in a single software-defined process or in multiple processes. At present, all modern supercomputers can be included in this class. The structure of this class is shown in Figure 2.1.

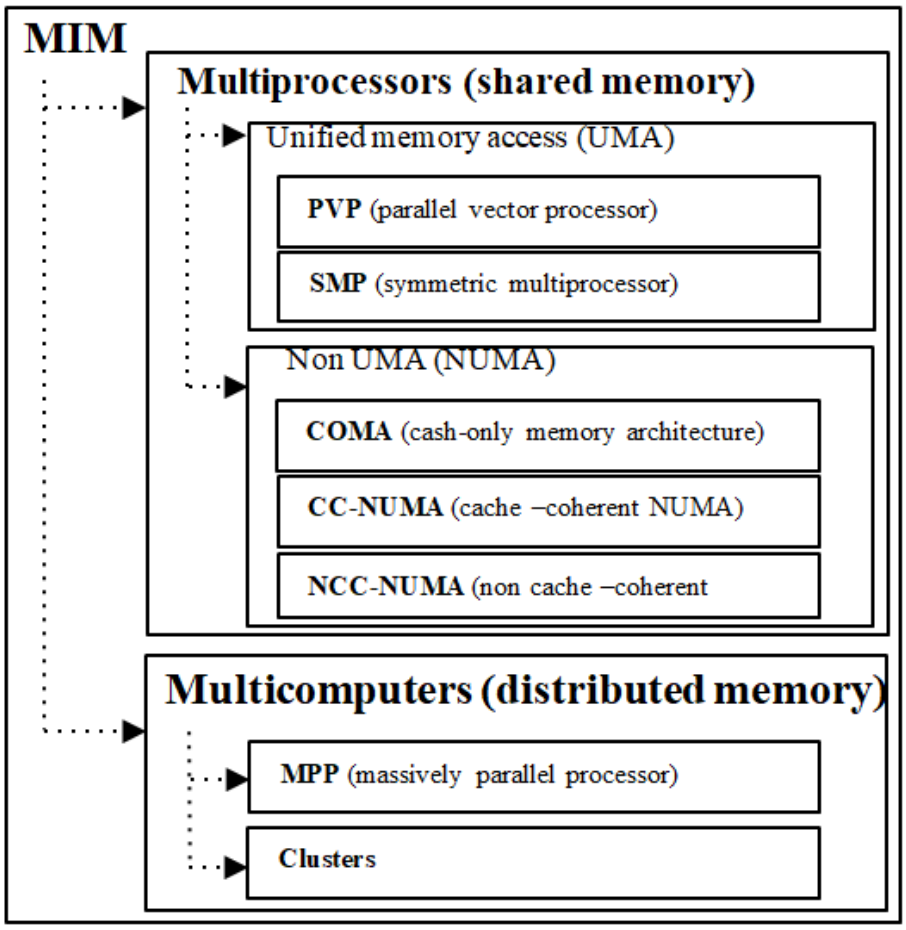


Figure 2.1 – MIMD structure

A MIMD computer has n processors, n command streams, and t data streams. Each processor works with its own command flow.

The machines have several processors that work asynchronously and independently. At any given time, different processors can execute different instructions on different pieces of data. The MIMD architecture can be used in a variety of applications, such as CAD / CAM, modeling, and communication connectors.

The MIMD class is very broad and combines many different types of architecture along with many computers.

MIMD systems can be divided into two subclasses: shared memory systems and distributed memory systems. For the first type of system, it is typical for any processor to have direct access to any of these common RAM slots. Allocated memory systems are usually a set of computer nodes. A node is understood as a stand-alone processor with independent RAM. In these systems, no processor can voluntarily access the memory of another processor.

Amdahl's Law

In the theory of parallel computing, there are some fundamental laws that limit the results of parallelization. One such law is Amdahl's law proposed by Gene Amdahl in 1967. This law provides for the acceleration of one program in n processors based on the prediction of the linear acceleration of the parallel part. For instance, any parallel program consists of a non-parallelizable serial part and a parallel part, if we denote the parallel part by p, then the serial part is $1 - p$. In this case, the maximum acceleration of parallelism in n processor $S(n)$ is as follows

$$S(n) = \frac{1}{(1-p) + p/n}$$

As n increases, the speedup becomes $1/(1-p)$.

Acceleration is limited by the total computing time of the serial part of the program. For 10-hour calculations, if the 8-hour calculations are parallelizable and the 2-hour calculations are non-parallelizable, then the maximum acceleration does not exceed 5 times.

Gustafson's law

Amdal's approach focuses on a constant-dimensional calculation because it deals with a code that takes a constant amount of sequential calculation time.

John Gustafson, a researcher at NASA Ames Research, takes a different view: mass parallel machines allow for calculations that have not been performed before because they allow calculations to be performed on a very large set of data over a period of time. In other words, the parallel platform not only speeds up the execution of the code: it allows you to solve large problems. Thus, he concluded that the increase in the total volume of the program was mainly due to the parallel part. Therefore, it is concluded that the acceleration can be increased, as its own share is small when the chain part of the total volume of the enlarged program remains unchanged.

Gustafson's law can be formulated as follows:

$$S\,(n) \;=\; 1 \;-\; p \;+\; pn$$

where:

S - theoretical acceleration of fulfillment of all tasks;

p - share of parallel calculations; respectively, $1 - p$ - share of serial calculations.

Gustafson's law amended the limits of Amdal's law for fixed-volume calculations, which do not change with the improvement of resources. Gustafson's law shows that programmers tend to choose the amount of computations that make full use of the resources of advanced resources - the faster the hardware, the greater the computational volume, while maintaining the same execution time.

According to Gustafson's law, efficiency rises to a new level, as the large volume of the problem, which is often solved, allows to increase the volume of parallel calculations.

The purpose of optimization using Amdal's law is to ensure the rapid operation of the program with a constant workload, and Gustafson's law - to perform a complex program without increasing the total execution time. This is the main difference between the two laws.

2.2 MPI PARALLEL IMPLEMENT OF 2 WAVE EQUATION WITH A DISTRIBUTIONAL COEFFICIENT

In this section we will discuss the numerical solution of the two-dimensional wave equation with a distributional coefficient by the implicit difference method. The approximation of the solution function is calculated at discrete points in the spatial grid, based on discrete time steps. The initial values are given by the initial value condition. First we will explain how to transform a differential equation into a finite-difference equation, respectively, a set of finite-difference equations that can be used to calculate an approximate solution. Then we will change this algorithm to parallelize this task on several processors. Special focus is on improving the performance of the parallel algorithm on different hardware platforms. In addition, we will run the implemented algorithm on a cluster and calculate the acceleration based on the execution time from 1 to 60 processors.

2.2.1 Introduction

The application of high-performance parallel computing in mathematical modeling opens up new possibilities for studying physical processes in longer time and more extensive spatial domains. Currently, various high-performance parallel computing is used in many areas. One of such applications is acoustics. One of the most important tasks of acoustics is the problem of wave field modeling.

There is a large amount of work devoted to numerical methods developed for the study of wave processes in recent decades. It includes a finite-difference method [25], a finite-volume method [22], a finite-element method [26], a two-level compact ADI method [29], an implicit Finite Difference Time Domain Methods [21], a boundary [integral] element method [29], and spectral methods [39]. A completely non-linear model must be applied to many problems. Most models have been developed for technical applications. These numerical methods provide some of the most natural methods for modeling the propagation and scattering of underlying waves in electromagnetic, acoustic and elastic studies. However, as indicated in [28], the aforementioned methods have several disadvantages if the second-order equations are converted to first-order systems before discretization, especially in the presence of several spatial dimensions. Therefore, recently, much attention has been paid to the

development of efficient finite-difference methods that directly discretize second-order differential equations [23, 24].

The two-dimensional approach (as a conformal method) considers a highly idealized wave field, since even monochromatic waves in the presence of side perturbations quickly acquire a two-dimensional structure. The difficulties encountered are not a direct result of the increase in size. The main complication is that the problem cannot be reduced to a two-dimensional problem, and even for the case of a two-periodic wave field, the problem of solving the Laplace-type equation for the velocity potential arises. Most models designed to study the three-dimensional dynamics of waves are based on simplified equations, such as second-order perturbation methods, in which higher-order terms are ignored. In general, it is unclear what effects are missing in such simplified models. Our current work is motivated by recent interest in the development and application of high-order compact difference methods for solving partial differential equations. Obviously, higher-order compact difference schemes have better resolution on stencils with a compact grid compared to non-compact or low-level methods [19, 30].

For multidimensional problems, the efficiency of an implicit compact difference scheme depends on the computational efficiency of the corresponding matrix solvers. From this point of view, the ADI method [40] is promising because they can decompose a multidimensional problem into a series of one-dimensional problems. It has been shown that schemes acquired are unconditionally stable.

For the proper assignment of large domains of modeling, two- or three-dimensional computational grids with a sufficient number of nodes are used. Calculations on such grids require more CPU time and computer memory resources. To accelerate the computation process, MPI technology was used in this paper, which allows the program to operate on larger grids[84, 85].

Here we consider some issues in the numerical simulation of some problems in the propagation of waves in acoustic on high performance computing systems.

2.2.2 Wave equation: theoretical part

We consider two-dimensional wave equation with a distributional coefficient

$$\frac{\partial^2 u}{\partial t^2} - a(t)\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = 0,\ (t,x,y) \in [0;T] \times [0;l] \times [0;l], \qquad (2.1)$$

subject to the initial conditions

$$u(0,x,y) = u_0(x,y),\ x,y \in [0,l], \qquad (2.2)$$

$$\frac{\partial u(0,x,y)}{\partial t} = 0,\ x,y \in [0,l], \qquad (2.3)$$

and boundary conditions

$$u(t,x,0) = 0,\ u(t,x,l) = 0,\ t \in [0,T],\ x \in [0,l], \qquad (2.4)$$

$$u(t,0,y) = 0,\ u(t,l,y) = 0,\ t \in [0,T],\ y \in [0,l], \qquad (2.5)$$

As it was theoretically outlined in [6, 7, 18] (also, see [31, 32, 33]), we start to analyse our problem by regularising distribution function $a(t)$ by a parameter ε, that is, we get

$$a_\varepsilon(t) := (a * \varphi_\varepsilon)(t),$$

as the convolution with the mollifier

$$\varphi_\varepsilon(t) = \frac{1}{\varepsilon}\varphi(t/\varepsilon),$$

with

$$\varphi(t) = \begin{cases} \frac{1}{c} e^{1/(t^2+1)}, |t| \le 1, \\ 0, |t| > 1, \end{cases}$$

where $c \simeq 0.443994$ to obtain $\int_{-1}^{1} \varphi(t)dt = 1$.

Hereinafter, we assume that $l = 10$. Then, instead of (2.1) we consider the regularised equation

$$\partial_{tt}^2 u_\varepsilon(t,x,y) - a_\varepsilon(t)\left(\partial_{xx}^2 + \partial_{yy}^2\right)u_\varepsilon(t,x,y) = 0,\ (t,x,y) \in [0,T] \times [0,10] \times [0,10]. \quad (2.6)$$

Here, we put

$$u_0(x,y) = \begin{cases} e^{1/((x-5)^2+(y-5)^2-0.5)}, & (x-5)^2+(y-5)^2-0.5 < 0, \\ 0, & (x-5)^2+(y-5)^2-0.5 \ge 0. \end{cases} \quad (2.7)$$

We note that $supp\ u_0 \subset [4.5,5.5] \times [4.5,5.5]$.

Numerical experiments

We introduce a space-time grid with steps h_1, h_2, τrespectively, in the variables x, y, t:

$$\omega_{h_1,h_2}^{\tau} = \{x_i = ih_1, i = \underline{0,N}; y_j = jh_2, j = \underline{0,N}; t^k = k\tau, k = 0,1,2 \ldots T/tau\} \quad (2.8)$$

and on this grid we will approximate the differential problem (2.6) with the conditions (2.2)–(2.5) using the finite difference method.

2.2.3 Alternating direction implicit (ADI) method

The ADI (Alternating Direction Implicit) method is a finite difference scheme and has long been used to solve partial differential equations (PDEs) in higher dimensions. Originally it was introduced by Peaceman and Rachford [37], but many variants have been invented throughout the years [38, 39, 40]. The ADI method uses only implicit finite difference operators, which makes it absolutely stable in problems that do not contain mixed derivatives. It has a fairly significant stability margin in problems with mixed derivatives. For the problem (2.2)–(2.6) the ADI method has the form

$$\frac{u_{i,j}^{k+1/2}-2u_{i,j}^{k}+u_{i,j}^{k-1/2}}{\tau^2} - a^{k+1/2}\left(\frac{u_{i+1,j}^{k+1/2}-2u_{i,j}^{k+1/2}+u_{i-1,j}^{k+1/2}}{h_1^2}\right) = 0, \qquad (2.9)$$

$$\frac{u_{i,j}^{k+1}-2u_{i,j}^{k+1/2}+u_{i,j}^{k}}{\tau^2} - a^{k+1}\left(\frac{u_{i,j+1}^{k+1}-2u_{i,j}^{k+1}+u_{i,j-1}^{k+1}}{h_2^2}\right) = 0. \qquad (2.10)$$

Using the implicit subscheme (2.9), the Thomas method is performed in the x direction, with the result that we get the grid function $u_{i,j}^{k+1/2}$. In the second fractional time step, using the subscheme (2.10), the Thomas method is performed in the direction of the y axis, with the result that we get the grid function $u_{i,j}^{k+1}$. The ADI method has the order $O(\tau^2 + h^2)$. In the following, we demonstrate numerical simulations. All calculations are made in C++ by using the Thomas method. For all simulations $tau = 0.05, h_1 = h_2 = 0.1$. In all visualization of result, we use Matlab R2018b.

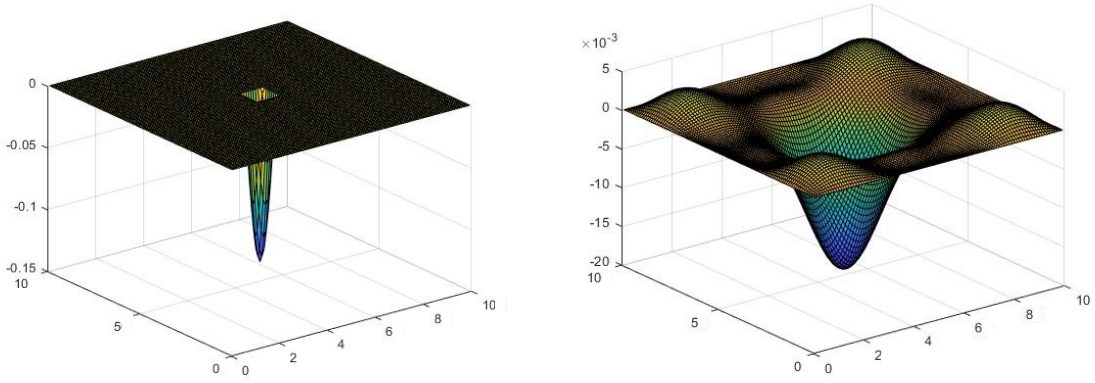


Figure 2.2 – In the left plot the graphic of the initial function u_0 is given. The solution of the problem u_ε at $t = 5$

2.2.4 Parallel Algorithm

On the basis of these methods we wrote the code of parallel modeling for the two-dimensional wave equation with distribution coefficient. The overlapping is based on the way divisions of domains and keeps the implicit nature of the scheme. It corresponds to a completely implicit interpretation of Yanenko [35]. Each processor processes one subdomain. On each temporary step all processors update the internal knots (in parallel), and then exchange the interface knots with the neighbors. Unlike methods of decomposition of areas for elliptic tasks, our procedure is algebraically equivalent to consecutive. Our strategy leads naturally to a message passing implementation. We have chosen to use MPI in order to evaluate its ease of use and expressiveness, and also to benefit from its portability. In accordance with the data decomposition format, Fig 2 shows the pure MPI parallelization which is designed on the basis of a sequential algorithm by using peer-to-peer mode and standard communication mode[41]. In another word, the parallel program is executed on a multiprocessor cluster by creating one MPI process for each CPU on the system, and each process deals with different data

obtained from the matrix. In this case all process interactions will happen via message-passing.

The message passing interface (MPI) is a standardized and portable programming interface for exchanging messages between multiple processors executing a parallel program in distributed memory. MPI works well on a wide variety of distributed storage architectures and is ideal for individual computers and clusters. However, MPI depends on explicit communication between parallel processes which requires mesh decomposition in advance due to data decomposition.

Therefore, MPI can cause load balancing and consume extra time. Our implementation uses MPICH2 because it is a freely accessible, compact implementation of MPI, a standard for message-passing for distributed-memory applications used in parallel computing[17]. MPICH is Free Software and is available for most flavors of UNIX and Microsoft Windows. MPI is standardized on many levels therefore it provides advantage for the user. For example you will be sure that your MPI code is executed in any MPI implementation launching on your architecture, even if the syntax has been standardized. Because the functional behavior of MPI calls is also standardized, your MPI calls should behave the same whatever of implementation, which ensures portability of your parallel programs

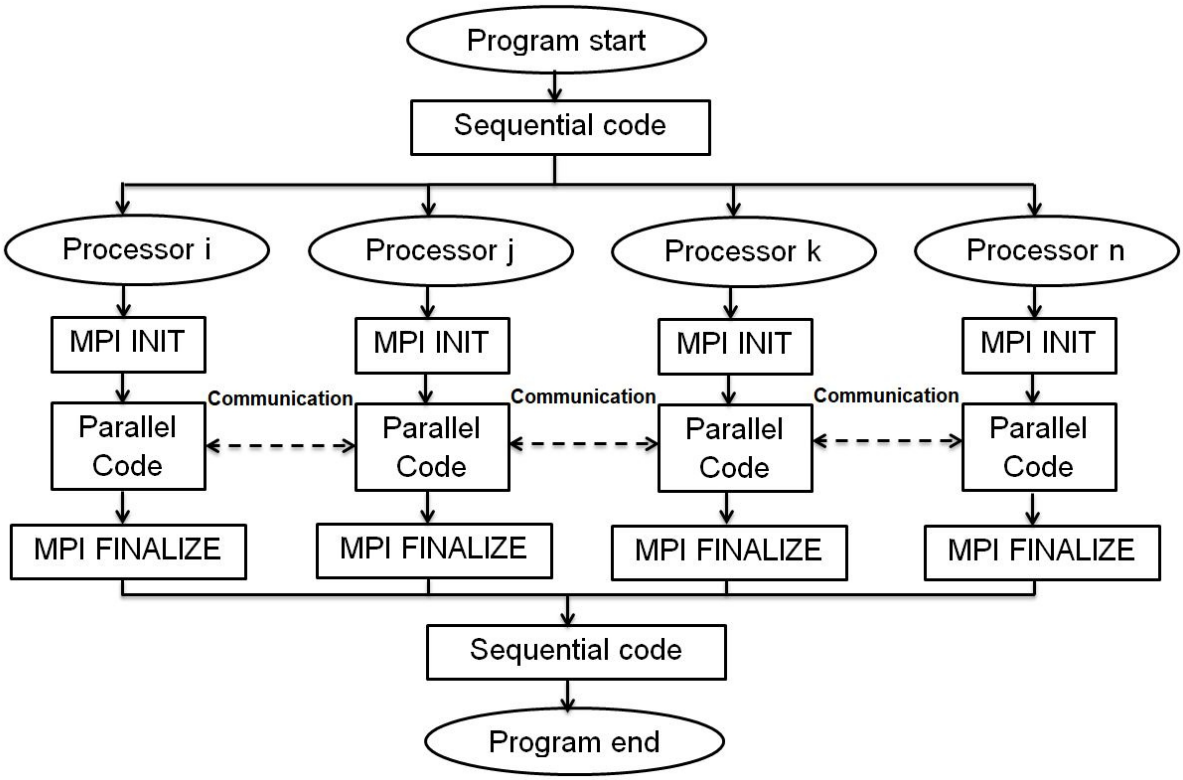


Figure 2.3 – Parallel model of MPI

To parallelize the problem (2.2)–(2.6) we apply the method proposed by N.N. Yanenko with coauthors [35]. The parallel algorithm is given in algorithm 1and the YanekoMethod (*a, *b, *c, *f, *y, n) function code in github [83].

Yanenko method

The Yanenko method can be applied not only to solve a set of systems of three-point equations, but also to solve a single system (introduced by A.A. Fedorov and A.N. Bykov in [42]). Consider a system of linear equations with a three-diagonal matrix of the following form:

$$a_i y_{i-1} + b_i y_i + c_i y_{i+1} = f_i, b_i \neq 0, i = 1,2, \dots N-1, b_0 y_0 + c_0 y_1 = f_0, a_N y_{N-1} + b_N y_N = f_N \quad (2.11)$$

For simplicity, let each processor have the same number of points $m = K/M$, where K is the number of unknowns (in our case, $K = N + 1$), M is the number of processors, the indexing will be global. Thus, on a separate processor with number j there will be only a part of the equations of system (2.11) with numbers from $(j-1) * m + 1$ to $j * m$, where j is the number of the processor. Denote y_{j*m} by z_j, $j = 0, \dots, M$ and look for solutions of system (4.1) in the form

$$y_{(j-1)*m+i} = z_{j-1} u_i + z_j v_i + w_i, i = 1, \dots m-1, j = 1, \dots M \quad (2.12)$$

where u, v, w are solutions of the following systems of equations:

$$a_i u_{i-1} + b_i u_i + c_i u_{i+1} = 0,\ u_{(j-1)*m} = 1, u_{j*m} = 0;\ a_i v_{i-1} + b_i v_i + c_i v_{i+1} = 0, i = (j-1) * m + 1, \dots j * m - 1,\ v_{(j-1)*m} = 0, v_{j*m} = 1; j = 1, \dots M\ a_i w_{i-1} + b_i w_i + c_i w_{i+1} = f_i,\ w_{(j-1)*m} = 0, w_{j*m} = 0; \quad (2.13)$$

Solutions of these three systems can be found by the Thomas method, and independently on each processor. We will call them prejudices, and this stage of solving the problem - the stage of finding prejudices.

In the equations with the numbers $j * m$ from the system (2.11):

$a_{j*m} y_{j*m-1} + b_{j*m} y_{j*m} + c_{j*m} y_{j*m+1} = f_{j*m}, j = 0, \dots M$, substitute the y combination (2.12). Thus, we obtain a system of three-point equations for finding z_j, which has the following form:

$$A_j z_{j-1} + B_j z_j + C_j z_{j+1} = F_j, j = 1, \dots M-1,\ B_0 z_0 + C_0 z_1 = F_0, A_M z_{M-1} + B_M z_M = F_M \quad (2.14)$$

with coefficients:

$$B_0 = b_0 + c_0 u_1, C_0 = c_0 v_1, F_0 = f_0 - c_0 w_1, A_j = a_{j*m} u_{j*m-1},$$
$$B_j = a_{j*m} v_{j*m-1} + b_{j*m} + c_{j*m} u_{j*m+1}, C_j = c_{j*m} v_{j*m+1},$$
$$F_j = f_{j*m} - a_{j*m} w_{j*m-1} - c_{j*m} w_{j*m+1}, j = 1, \dots M-1,$$
$$A_M = a_{M*m} u_{M*m-1}, B_M = b_{M*m} + a_{M*m} v_{M*m-1}, F_M = f_{M*m} - a_{M*m} w_{M*m-1}.$$

Let's call this stage - the stage of finding the boundary-processor solutions. The dimension of this system of equations is equal to the number of processors, which is

significantly less than the number of problem points. At the last stage, we restore the final decision using formula (2.12). So, the method of Yanenko contains three stages:

1) finding prejudices,
2) finding the boundary-processor solutions,
3) restore solution.

The first and third stages are performed independently on each device (universal processor, accelerator or coprocessor). And the second stage requires communication between MPI processes. Thus, the efficiency of the parallelization of the Yanenko method directly depends on the efficiency of the parallelization of this stage.

Parallel algorithm

1: init parallel environment
2: for all MPI processes do in parallel
3: get the input parameters N, L, pi,T
4: allocate local memory $u_0, u, alfa, beta, U, V, W$
5: Initialize u_0, u_1 with initial values and boundary conditions
6: record time t1
7: **while** (t < T) **do**
8: **for** j ← 0 to N-1 **do**
9: **for** i ← 1 to N-1 **do**
10: **Compute** a_i, b_i, c_i, f_i
11: $a_i, b_i \leftarrow a\tau^2$
12: $c_i \leftarrow h^2 + 2a\tau^2$
13: $f_i \leftarrow u_{i,j}^k(-2h^2) + u_{i,j}^{k-1/2}$
14: $u_{i,j}^{k+1/2} \leftarrow YanenkoMethod(a_i, b_i, c_i, f_i, N)$
15: **for** i ← 0 to N-1 **do**
16: **for** j ← 1 to N-1 **do**
17: **Compute** a_j, b_j, c_j, f_j, N
18: $a_j, b_j \leftarrow a\tau^2$
19: $c_j \leftarrow h^2 + 2a\tau^2$
20: $f_j \leftarrow u_{i,j}^{k+1/2}(-2h^2) + u_{i,j}^k(h^2)$
21: $u_{i,j}^{k+1} \leftarrow YanenkoMethod(a_j, b_j, c_j, f_j, N)$
22: $swap(u_{i,j}^{k-1/2}, u_{i,j}^{k+1/2})$
23: $swap(u_{i,j}^k, u_{i,j}^{k+1})$
24: $t \leftarrow t + \tau$
25: **endwhile**
26: record time t2
27: output t2- t1
28: stop parallel environment

2.2.5 Experimental Results

All calculations and tests were carried out on a personal computer core i7 8th generation RAM 8GB and on a 60 core cluster consisting of 5 nodes, connected by a high-speed InfiniBand HCA 40Gb/s, QDR bus. Each node contains two Intel Xeon E5-2620v2 6C/12T 2.10GHz 15MB processors and 8 GB of RAM. We used the personal computer for sequential calculations and visualizing results in Matlab programs. The

performance of a parallel algorithm is determined by calculating its speedup. Strong scaling speedup is defined as the ratio of the best-case execution time of the sequential algorithm for a particular problem to the worst-case execution time of the parallel algorithm.

$$S = \frac{T_s}{T_p};$$

where T_s is the computational time for running the program using one processor, T_p is the computational time running the same program with p processors. The efficiency is defined as the ratio of speedup to the number of processors. Efficiency measures the fraction of time for which a processor is usefully utilized.

$$E = \frac{S}{p} = \frac{T_s}{pT_p};$$

where S is the speedup, p is the number of processors. The test results are presented in Figure 2.4.

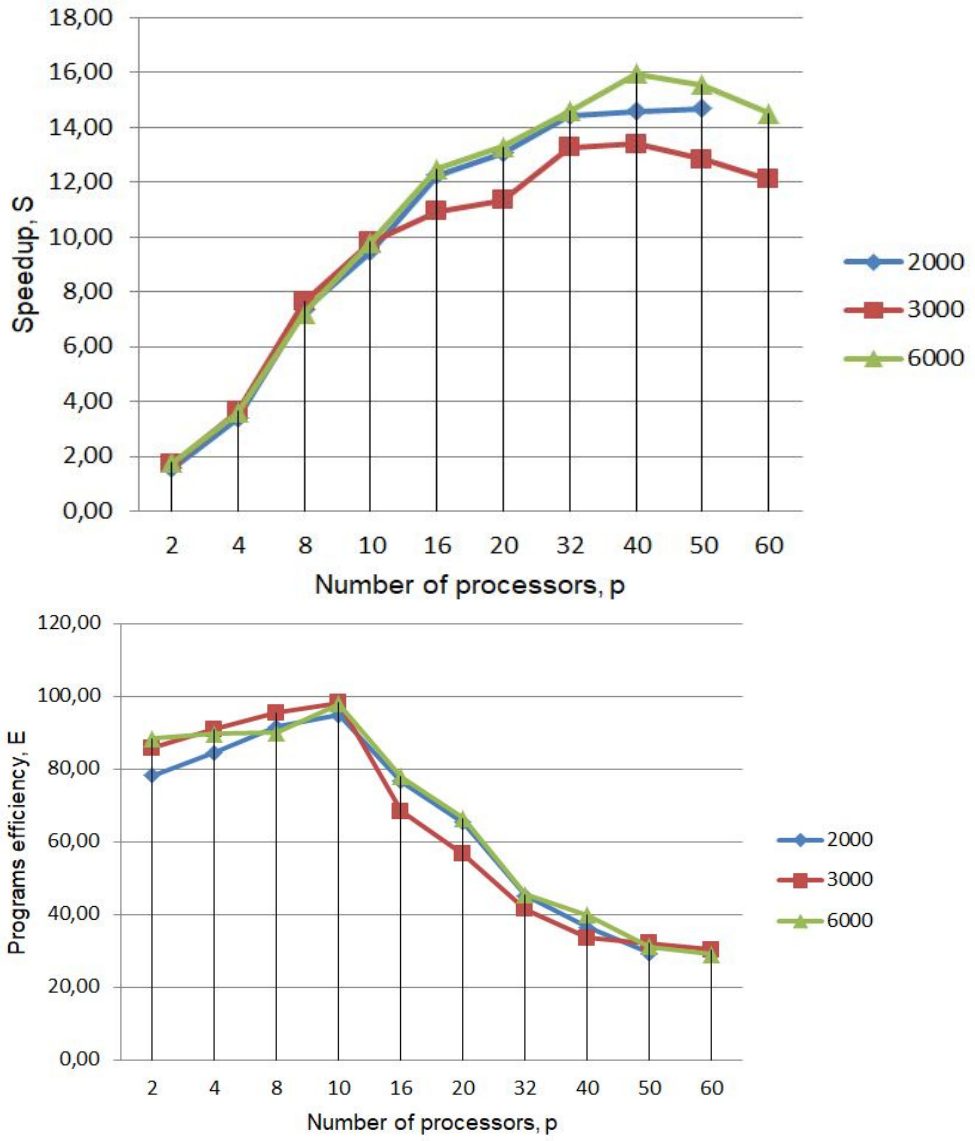


Figure 2.4 – Speedup and Programs efficiency (%)

The small cluster that we launched the tasks consists of 5 nodes, and each node consists of 12 physical cores. These two figures from the calculation results, shows that the number of cores up to 10 the speedup and efficiency improves significantly, since the number of cores exceeds 10 speedup slowly rises and while the efficiency decreases, as the grid size increases, for example N = 3000 and N = 6000, the efficiency is significantly reduced, because the time spent on exchanging information between the cores is quite large that the InfiniBand cable throughput for transmitting information between supercomputers is lower than that of the motherboard information bus whom, therefore, the effectiveness is reduced. Namely the method proposed by Yanenko to parallelize the Thomas algorithm would be effective only if the exchange of data between the cores is very high.

We have implemented a parallel solver for the 2D wave equation with a distributional coefficient, using the MPI standard.

The implementation of the parallelization method of the Thomas algorithm for solving the wave equation arising in the simulation of two-dimensional physical processes on supercomputers has been implemented. The test results show that the method of Yanenko is much more effective on individual nodes than a several nodes. This means that this method is more effective on personal computers than a cluster this is because it takes a lot of time to exchange data between nodes.

In the future, it is planned to try to apply the method of parallel-cyclic reduction at the second stage of the Yanenko method, to investigate the possibility of increasing the parallelization efficiency, and to use vectorization in order to increase the acceleration from using coprocessors.

2.3 GPU computing for time depending 2D tsunami wave equation

In this section, we present the numerical implementations of the 2D tsunami wave equation. We considered a simulation of tsunami wave propagation in the shallow water area of the coast that was generated by an underwater earthquake using an implicit finite difference scheme. A parallelization algorithm on GPU is given. We carry out a special focus on improving the performance of the parallel algorithms. It is observed that the codes running on the CUDA platform give expected results. By comparing our tests on the GPU to those obtained by running the serial code of the same simulation on the CPU, GPU simulations are found to run quite faster than ones run on the CPU.

The tsunami wave equation is one of the fundamental equations in many engineering and physical science problems. In this paper we consider numerical simulations of the Cauchy problem for the two-dimensional tsunami equation

$$\frac{\partial^2 u}{\partial t^2} - \left[\frac{\partial}{\partial x}\left(H(t,x,y)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(H(t,x,y)\frac{\partial u}{\partial y}\right)\right] + \frac{\partial^2 H}{\partial t^2} = f(t,x,y), \quad (t,x,y) \in (0,T)\times\Omega, \tag{2.15}$$

where $\Omega = (0,X) \times (0,Y)$ for some fixed $X, Y > 0$. The initial values are given by the initial value condition

$$u(0,x,y) = \varphi(x,y), \ \frac{\partial u}{\partial t}(0,x,y) = \psi(x,y), \ (x,y) \in \underline{\Omega}, \tag{2.16}$$

where $\underline{\Omega} = (0,X) \times (0,Y)$. Since the domain Ω is bounded we require some conditions on its boundary, for example, the Dirichlet boundary condition

$$u(t,x,y)|_{\partial\Omega} = g(t,x,y), \ t \in [0,T]. \tag{2.17}$$

where H(t,x, y) is the still-water depth (typically obtained from an electronic map). The t-dependence in H gives a moving bottom to model, such as an underwater slide or earthquake. In this work, we consider the more smooth singularity case as shown below

$H(t,x,y) = H_0(x,y) + \theta(t)H_1(x,y)$

this means a specific place of water bottom go up during a period of time t_0 and t_1

$$H_0(x,y) = \begin{cases} 0.5x, & 0 \le x < 20 \\ 10 + 2.25(x-20), & 20 \le x < 60 \\ 100, & 60 \le x < 100 \end{cases} \tag{2.18}$$

$$H_1(x,y) = \begin{cases} -10, & 61 \le x < 71, 45 \le y < 75, \\ 0, & else \end{cases} \tag{2.19}$$

here $\theta(t) = |\theta_2(t) - \theta_1(t)|$

$$\theta_1(t) = \{a, \ t \ge t_0, 0, \ else \tag{2.20}$$

$$\theta_2(t) = \{a, \ t < t_1, 0, \ else \tag{2.21}$$

in our case $H_{tt} \approx \phi(t)H_1(x,y)$ where $\phi(t) = \phi 1_{\varepsilon}^{'}(t) - \phi 2_{\varepsilon}^{'}(t)$

$$\phi 1_{\varepsilon}'(t) = \begin{cases} \frac{2(t-t_0)\varepsilon}{C((t-t_0)^2-\varepsilon^2))^2} \exp\left(\frac{\varepsilon^2}{(t-t_0)^2-\varepsilon^2}\right), & t_0 - \varepsilon < t < t_0 + \varepsilon, \\ 0, \ \text{in other cases} \end{cases} \tag{2.22}$$

$$\phi 2_{\varepsilon}'(t) = \begin{cases} \frac{2(t-t_1)\varepsilon}{C((t-t_1)^2-\varepsilon^2))^2} \exp\left(\frac{\varepsilon^2}{(t-t_1)^2-\varepsilon^2}\right), & t_1 - \varepsilon < t < t_1 + \varepsilon, \\ 0, \ \text{in other cases} \end{cases} \tag{2.23}$$

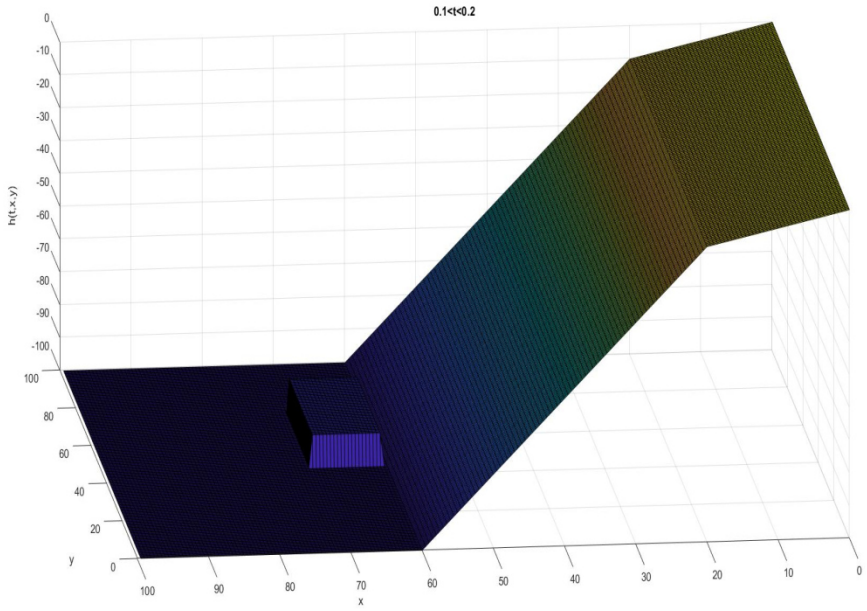


Figure 2.5 – $H_0(x,y)$-water bottom and $H_1(x,y)$ - place where earthquakes take place

In this work we are interested in the singular cases of the function h, source term f, initial functions φ and ψ. To deal with the theoretical part, Ruzhansky and his co-authors introduced so-called "very weak solutions" in [4, 6, 31, 32].

Here, we offer a finite-difference equation to calculate an approximate solution of the boundary value problem (2.15)–(2.17). It is a subject to parallelize on GPU the standard algorithms on CPU. Special focus is on improving the performance of parallel computing. In addition, we run the implemented parallel code on the GPU and serial codes on the central processor to calculate the acceleration based on the execution time. We find out that the parallel code that runs on the GPU gives the expected results by comparing our results to those obtained by running the serial code of the same simulation on the CPU.

2.3.1 Preliminaries

We introduce space-time grids with steps τ, h_1, h_2 in the variables t, x, y, respectively, that are

$$\omega^{\tau}_{h_1,h_2} = \{(t_k, x_i, y_j): t_k = k\tau; x_i = ih_1; y_j = jh_2,\ \ (k,i,j) \in I\}, \Omega^{\tau}_{h_1,h_2} = \{(t_k, x_i, y_j): t_k = k\tau; x_i = ih_1; y_j = jh_2,\ \ (k,i,j) \in \underline{I}\}, \quad (2.24)$$

where

$$I := \{(k,i,j) \in Z^3_+: 0 < k \le M; 0 < i < N_1; 0 < j < N_2\},$$

$$\underline{I} := \{(k,i,j) \in Z^3_+: 0 \le k \le M; 0 \le i \le N_1; 0 \le j \le N_2\},$$

and

$$X = h_1 N_1, Y = h_2 N_2, T = \tau M.$$

One calculates an approximate solution from discrete points in the time-spatial grid $\Omega^{\tau}_{h_1,h_2}$. On this grid we approximate the problem (2.15)–(2.17) using the finite difference method. For simplicity, we put $N := N_1 = N_2$ and denote $h := h_1 = h_2$. Consider the Crank-Nicolson scheme for the problem (2.15)–(2.17). First we transform the partial differential equation (2.15) into the implicit finite-difference equation

$$\frac{u_{i,j}^{k+1}-2u_{i,j}^{k}+u_{i,j}^{k-1}}{\tau^2} - \frac{H_{i,j}^{k}}{4h^2}((u_{i+1,j}^{k+1} - 2u_{i,j}^{k+1} + u_{i-1,j}^{k+1} + u_{i+1,j}^{k-1} - 2u_{i,j}^{k-1} + u_{i-1,j}^{k-1}) + (u_{i,j+1}^{k+1} - 2u_{i,j}^{k+1} + u_{i,j-1}^{k+1} + u_{i,j+1}^{k-1} - 2u_{i,j}^{k-1} + u_{i,j-1}^{k-1})) + (\frac{H_{i+1,j}^{k}-H_{i,j}^{k}}{h})(\frac{u_{i+1,j}^{k}-u_{i,j}^{k}}{h}) + (\frac{H_{i,j+1}^{k}-H_{i,j}^{k}}{h})(\frac{u_{i,j+1}^{k}-u_{i,j}^{k}}{h}) = f_{i,j}^{k} \quad (2.25)$$

for $(k,i,j) \in \omega^{\tau}_{h_1,h_2}$, where $h_{i,j}^{k} := h(ih, jh)$, $f_{i,j}^{k} := f(k\tau, ih, jh)$ with initial conditions

$$u_{i,j}^{0} = \varphi_{i,j}, \; u_{i,j}^{1} - u_{i,j}^{0} = \tau\varphi_{i,j}, \quad (2.26)$$

for $(i,j) \in \underline{0,N} \times \underline{0,N}$, and with boundary conditions

$$u_{0,j}^{k} = 0, \; u_{N,j}^{k} = 0, \; u_{i,0}^{k} = 0, \; u_{i,N}^{k} = 0, \quad (2.27)$$

for $(j,k) \in \underline{0,N} \times \underline{0,M}$ and $(i,k) \in \underline{0,N} \times \underline{0,M}$, respectively.

It is well-known, that the implicit scheme (2.25)–(2.27) is unconditionally stable and it has accuracy order $O(\tau + |h|^2)$, see, for example [16].

We solve the difference equation (2.25) by the alternating direction implicit (ADI) method, namely, dividing it into two sub problems

$$\frac{u_{i,j}^{k+1/2}-2u_{i,j}^{k}+u_{i,j}^{k-1/2}}{\tau^2} - \frac{H_{i,j}}{4h^2}((u_{i+1,j}^{k+1/2} - 2u_{i,j}^{k+1/2} + u_{i-1,j}^{k+1/2}) + (u_{i+1,j}^{k-1/2} - 2u_{i,j}^{k-1/2} + u_{i-1,j}^{k-1/2})) - (\frac{H_{i+1,j}-h_{i,j}}{h})(\frac{u_{i+1,j}^{k-1/2}-u_{i-1,j}^{k-1/2}}{2h}) = \frac{f_{i,j}^{k-1/2}}{2}, \quad (2.28)$$

and

$$\frac{u_{i,j}^{k+1}-2u_{i,j}^{k+1/2}+u_{i,j}^{k}}{\tau^2} - \frac{H_{i,j}}{4h^2}((u_{i,j+1}^{k+1} - 2u_{i,j}^{k+1} + u_{i,j-1}^{k+1}) + (u_{i,j+1}^{k} - 2u_{i,j}^{k} + u_{i,j-1}^{k})) - (\frac{H_{i,j+1}-H_{i,j}}{h})(\frac{u_{i,j+1}^{k}-u_{i,j-1}^{k}}{2h}) = \frac{f_{i,j}^{k}}{2}. \quad (2.29)$$

Now, the equation (2.15) can be solved by the ADI method in two sub-steps. At the first sub-step, we solve the equation (2.28) in x direction:

$$\tilde{A}_i u_{i+1,j}^{k+1/2} + \tilde{B}_i u_{i,j}^{k+1/2} + \tilde{C}_i u_{i-1,j}^{k+1/2} = \tilde{F}_i, \quad (2.30)$$

where $\tilde{A}_i = -\tau^2 h_{i,j}$, $\tilde{B}_i = 2\tau^2 h_{i,j} + 4h^2$, $\tilde{C}_i = -\tau^2 h_{i,j}$ and $\tilde{F}_i = -8h^2 u_{i,j}^{k} + (4h^2 + 2\tau^2 h_{i,j})u_{i,j}^{k-1/2} - h_{i,j}\tau^2(u_{i+1,j}^{k-1/2} + u_{i-1,j}^{k-1/2}) - 2h\tau^2(u_{i+1,j}^{k-1/2} + u_{i-1,j}^{k-1/2})(h_{i+1,j} - h_{i,j})/h + 2h^2\tau^2 f_{i,j}^{k-1/2}$.

As the second sub-step, we solve the equation (2.29) in y direction:

$$\tilde{A}_j u_{i,j+1}^{k+1} + \tilde{B}_j u_{i,j}^{k+1} + \tilde{C}_j u_{i,j-1}^{k+1} = \tilde{F}_j, \quad (2.31)$$

where $\tilde{A}_j = -\tau^2 h_{i,j}$, $\tilde{B}_j = 2\tau^2 h_{i,j} + 4h^2$, $\tilde{C}_j = -\tau^2 h_{i,j}$ and $\tilde{F}_j = -8h^2 u_{i,j}^{k+1/2} + (4h^2 + 2\tau^2 h_{i,j})u_{i,j}^k - h_{i,j}\tau^2(u_{i+1,j}^k + u_{i-1,j}^k) - 2h\tau^2(u_{i+1,j}^k + u_{i-1,j}^k)(h_{i+1,j} - h_{i,j})/h + 2h^2\tau^2 f_{i,j}^k$.

Indeed, each of the equations (2.30) and (2.31) can be written in the matrix form $[A]U = f$ as follows

$$\begin{bmatrix} b_1 & c_1 & 0 & \cdots & \cdots & \cdots & 0 \\ a_1 & b_2 & \ddots & \ddots & & & \vdots \\ 0 & a_2 & \ddots & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & c_{n-2} & 0 \\ \vdots & & & \ddots & & b_{n-1} & c_{n-1} \\ 0 & \cdots & \cdots & \cdots & 0 & a_{n-1} & b_n \end{bmatrix} \times \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ u_n \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ f_n \end{pmatrix}$$

Here, this system can be further simplified and solved easily. However, the coefficients may vary at each time step for more complex applications. From a parallelization solver point of view, we will keep the notations of the original system for generality.

2.3.2 High-performance parallel computing

Already for several years, GPUs have been used to accelerate well parallelizable computing, only with the advent of a new generation of GPUs with multicore architecture, this direction began to give tangible results. The goal of this work is to develop a parallel implementation of the finite difference method for solving two-dimensional wave equations on a graphics processor using CUDA technology and to study the efficiency of parallelization by comparing the time of solving two-dimensional wave equations on a GPU and a central processor.

The graphics processing unit (GPU) is a highly parallel, multi-threaded, and multi-core processor with enormous processing power. Its low cost and high bandwidth floating point operations and memory access bandwidth are attracting more and more high performance computing researchers [44]. In addition, compared to cluster systems, which consist of several processors, computing on a GPU is inexpensive and requires low power consumption with equivalent performance. In many disciplines of science and technology, users were able to increase productivity by several orders of magnitude using graphics processors [45, 46, 86].

GPU programming on NVIDIA graphics cards has become significantly easier with the introduction at the end of 2006 of the CUDA programming language (NVIDIA Corporation 2009a), which is relatively easy to learn because its syntax is similar to C. With GPU becoming a viable alternative to CPU for parallel computing, aforementioned parallel tridiagonal solvers and other hybrid methods have been implemented on GPUs

[47] – [54]. Zhang et al. [47] first implemented parallel cyclic reduction (PCR) and then proposed a CR-PCR hybrid algorithm. A hybrid PCR-Thomas method was proposed by Sakharnykh [53], and it was also studied by Zhang et al. [47]. There are many examples in the literature of successfully using GPUs for wave propagation simulation [55] – [60].

Cyclic reduction method (CR)

Besides the Thomas method, other methods are also used to solve the system of linear algebraic equation (SLAE) of the tridiagonal matrix, which in practice is often more efficient. One of these methods is the cyclic reduction method. The main limitation of this method is that it works only in cases where the matrix has a dimension equal to the degree of two. Cyclic reduction method was invented by W. Hockney in 1965 [61] and the CR method consists of two steps: forward reduction and backward substitution. The forward reduction step sequentially eliminates the odd-indexed unknowns and then unknowns are reordered and the process is continued until one equation with one unknown is left. The backward substitution step solves the remaining one equation and finds the unknown y, consequently finds all unknowns from the previous steps. Each step consists of $log_2 n - 1$ sub-steps where n is the system size. In each step of forward reduction, we update all even-indexed equations with equation i of the current system as a linear combination of equations $i, i+1$ and $i-1$, so that we derive a system of only even-indexed unknowns. Equation i has the form $a_i y_{i-1} + b_i y_i + c_i y_{i+1} = f_i$. The updated values of a_i, b_i, c_i and f_i are

$$a_i' = -a_{i-1}q_1; b_i' = b_i - c_{i-1}q_1 - a_{i+1}q_2\, c_i' = -c_{i+1}q_2; f_i' = f_i - f_{i-1}q_1 - f_{i+1}q_2\, q_1 = \frac{a_i}{b_{i-1}}; q_2 = \frac{c_i}{b_{i+1}} \quad (2.32)$$

In each step of backward substitution, we solve all odd-indexed unknowns y_i in parallel by substituting the already solved y_{i-1} and y_{i+1} values to equation i,

$$y_i = \frac{f_i' - a_i' y_{i-1} - c_i' y_{i+1}}{b_i'} \quad (2.33)$$

Figure 2.6 shows the communication pattern of the algorithm for an 8-unknown system.

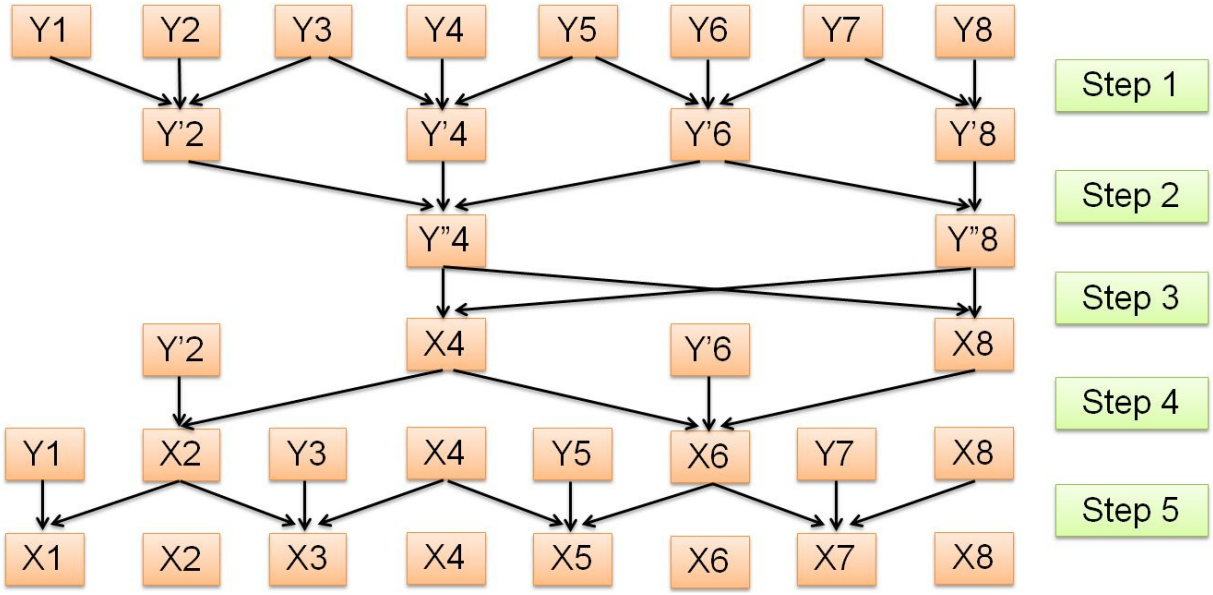


Figure 2.6 – Communication pattern for CR in the 8-unknown case, showing the dataflow between each equation, labeled $y1$ to $y8$. Letters y' and y'' stand for updated equation [47].

There are:
Step 1: forward reduction reduced to a 4 unknown system
Step 2: forward reduction reduced to a 2 unknown system
Step 3: solve 2 unknown system
Step 4: backward Substitution solves the remaining 2-unknowns
Step 5: backward Substitution solves the remaining 4-unknowns

The algorithm of the Cyclic reduction method is given below:

Algorithm 1 Sequential algorithm

```
1: function CRM_serial (*a, *b, *c, *f, *y, n)
2:     for(i = 0; i < log2(n + 1) - 1; i + +){
3:         for(j = 2^(i+1) - 1; j < n; j = j + 2^(i+1)){
4:             offs = 2^i;
5:             i1 = j - offs;
6:             q1 = a[j]/b[i1];
7:             q2 = c[j]/b[j];
8:             b[j] = b[j] - c[j - offs] * q1 - a[j+ offs] * q2;
9:             f[j] = f[j] - f[j - offs] * q1 - f[j + offs] * q2;
10:            a[j] = -a[j - offs] * q1;
11:            c[j] = -c[j + offs] * q2;
12:        }
13:    }
14:    k = (n - 1)/2;
15:    y[k] = f[k]/b[k];
16:    for(i = log2(n + 1) - 2; i >= 0; i - -){
17:        for(j = 2^(i+1) - 1; j < n; j = j + 2^(i+1)){
18:            offs = 2^i;
19:            i1 = j - offs;
20:            i2 = j + offs;
21:            if (j! = i1)
22:               y[i1] = (f[i1] - a[i1] * x[i1 - offs] -c[i1]* y[i1 + offs])/b[i1];
23:            if (j! = i2)
24:               y[i2] = (f[i2] - a[i2] * x[i2 - offs] -c[i2] * y[i2 + offs])/b[i2];
25:        }
26:    }
27: end function
```

Figure 2.7 – Sequential algorithm

Using the implicit subscheme (2.28), the cyclic reduction method is performed in the x direction, with the result that we get the grid function $u_{i,j}^{k+1/2}$. In the second fractional time step, using the subscheme (2.29), the Cyclic reduction method is performed in the direction of the y axis, with the result that we get the grid function $u_{i,j}^{k+1}$. The Cyclic reduction algorithm has the order $O(\tau + h^2)$, i.e. the first order in time and the second in x and y variables. In the following, we demonstrate numerical simulations. All calculations are made in C++ by using the Cyclic reduction algorithm. For all simulations $\Delta t = 0.05, \Delta x = \Delta y = 0.5$. In all visualization of result, we use Matlab R2018b.

For simulation we use initial condition:

$$U(x, y, 0) = 0$$

$U_t(x, y, 0) = 0$ and dirichlet boundary conditions. Some results are illustrated in Fig.2.8.

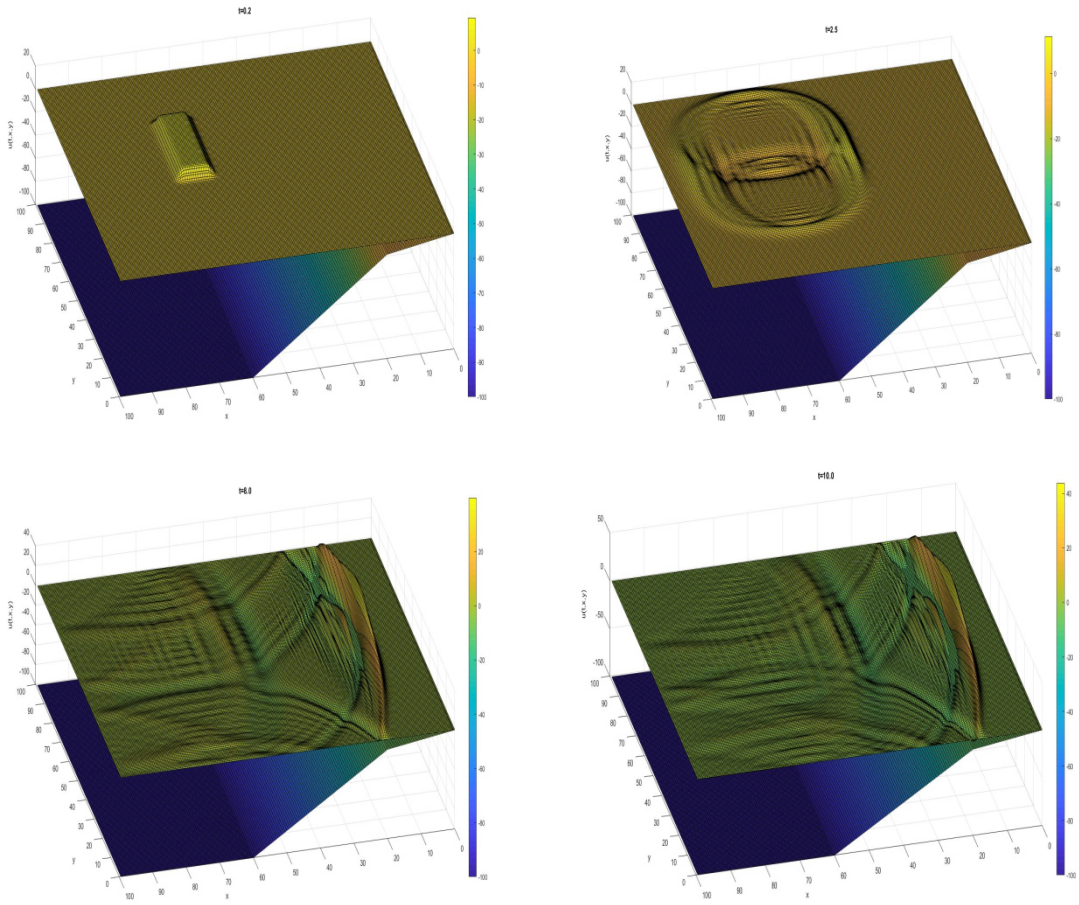


Figure 2.8 – The water bottom and displacement of the wave at different times

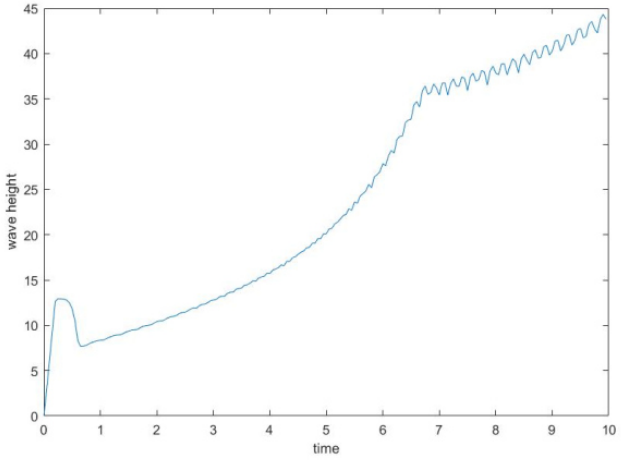


Figure 2.9 – Changes in wave height over time

Numerical Accuracy Analysis

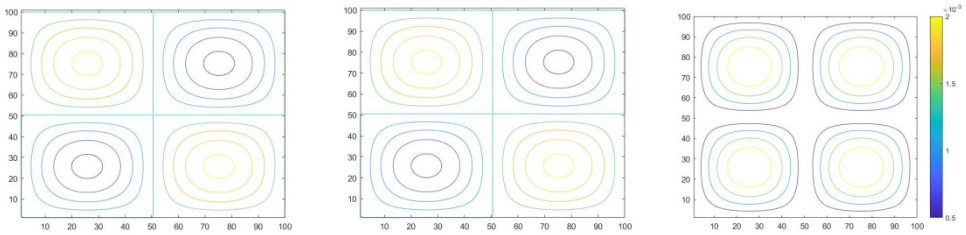
The numerical solution requires a certain level of numerical accuracy. We check the accuracy of we used the finite-difference scheme using absolute error and norm error L^2 which is defined as

$$err_{abs} = ||u - u_{num}|| \tag{2.34}$$

$$L^2 = \sqrt{\sum_{i=1}^{n}(u - u_{num})^2 \Delta x \Delta y} \tag{2.35}$$

where u denotes the exact value and u_a denotes the approximation.

We adopt the unit square $(x, y) \in [0,1] \times [0,1]$ as the spatial solution domain with 100 elements per each side and 100 interior points, $H(t, x, y) = 1$, with initial condition $u(x, y, 0) = sin(2\pi x)sin(2\pi y)$, $\frac{\partial u(x,y,0)}{\partial t} = 0$ and $u(0, y, t) = u(1, y, t) = u(x, 0, t) = u(y, 1, t) = 0$ on the boundaries. The analytical solution of equation (2.15) is as flows: $u(x, y, t) = cos(2\pi\sqrt{2}t)sin(2\pi x)sin(2\pi y)$.



a) b) c)

Figure 2.10 – Solution for time $t = 1, \Delta x = \Delta y = 0.01, \Delta t = 0.001$ a) exact, b) numerical, c) absolute error

2.3.3 Implementation on graphics cards using CUDA

Nowadays Graphics Processing Units(GPUs) or graphics processors have evolved from fixed-function processors specialized for three-dimensional graphics operations to a fully programmable computing platform for a wide variety of computationally demanding applications. Modern GPUs are massively data-parallel throughput-oriented many-core processors capable of providing TFLOPS of computing performance and quite high memory bandwidth compared to a high performance CPU. On the other hand, owing to their peculiar and particular architecture, developing an efficient algorithm that gives the high performance from the GPU, is a difficult task. Traditional algorithms developed for scalar architectures (e.g. CPU) do not translate naturally to parallel architectures (e.g. GPU)[62]. In this paper, we present an efficient parallel algorithm based on the CR method for numerically solving the 2D wave equation on the GPU.

GPU architecture

Latest NVIDIA's computing architecture such as NVIDIA GeForce RTX 2080Ti (Turing TU102 architecture [65]) which is used in this work is the powerful Turing architecture, innovative technology and 11 GB of next-generation ultra-fast memory GDDR6 make it the world's best graphics card, as illustrated in Fig.2.11. The GPU architecture consists of numerous cores called streaming processors (SPs), that are grouped in a set of streaming multiprocessors (SMs). Every SM processes instructions in a single-instruction multiple-threads (SIMT) mode and supports a multithreading execution mechanism. The threads in groups of 32 parallel threads called warp are created, managed, scheduled and executed by the SM. The instruction (fetch/decode), control and warp scheduling logic are shared across all the SPs on the SM. In an application a block represents a group of threads that can be executed serially or in parallel. During program execution, the hardware will appoint a whole block to one SM, although several blocks can execute in the same SM. When a SM executes one or more thread blocks, it divides them into warps and each warp will be scheduled by a warp scheduler for execution. one common instruction executed by a warp at the same time , that is why full efficiency is realized when all 32 threads of a warp agree on their execution path.

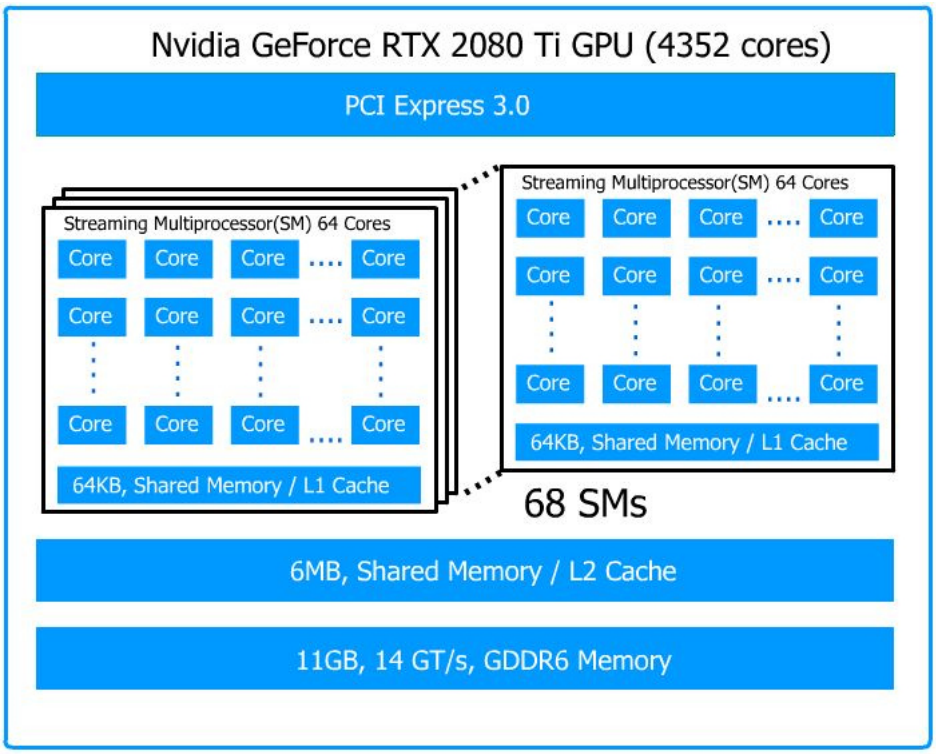


Figure 2.11 – Turing TU102 architecture

GPUs use areas of memory, such as global, local, constant, texture, shared, and registers. These areas of memory have different characteristics that reflect their different

uses in the application. Global, local, texture, and constant memory are located in DRAM (Dynamic Random Access Memory) with relatively large access latency[63].

CUDA implementation

In 2007, NVIDIA introduced CUDA, an extension to C programming language, for general purpose computing on graphics processors. It is designed so that its constructions allow a natural expression of concurrency at the data level. A CUDA program consists of two parts: a sequential program running on the CPU, and a parallel part running on the GPU [63, 64]. The parallel part is called the kernel. A C program using CUDA extensions hand out a large number of copies of the kernel into available multiprocessors to be performed contemporaneously. The CUDA code consists of three computational steps: transferring data to the global GPU memory, running the CUDA core, and transferring the results from the GPU to the CPU memory. We have designed a CUDA program based on cyclic reduction method (2.32) and (2.33). The algorithm for solving the problem (2.15) is shown in Algorithm as bellow.

Implementation of 2D tsunami wave equation

Algorithm 1 Implementation of 2D tsunami wave equation

```
compute initial condition matrix u0
from initial condition (1.2) we can get u1=u0
repeat
  for j ← 0 to n do
    for i ← 1 to n do
      calculate tridiagonal system elements a_i, b_i, c_i, f_i and copy host to device
      call function CR(a_i, b_i, c_i, f_i, y_i, n)
      copy result y_i from device to host Ux_{i,j} ← y_i
    end for
  end for
  for i ← 0 to n do
    for j ← 1 to n do
      calculate tridiagonal system elements a_j, b_j, c_j, f_j and copy host to device
      call function CR(a_j, b_j, c_j, f_j, y_j, n)
      copy result y_j from device to host Uy_{i,j} ← y_j
    end for
  end for
  swap(u1,Ux)
  swap(u0,Uy)
  t ← t + Δt
until t <= T
```

Here, $u0$, $u1$, Ux, Uy denote $u_{i,j}^{k-1/2}$, $u_{i,j}^{k}$, $u_{i,j}^{k+1/2}$, $u_{i,j}^{k+1}$ respectively. The calculation formula of a_i, b_i, c_i, f_i is shown 2.30, like that a_j, b_j, c_j, f_j is shown 2.31. The $CR()$ function includes 3 device functions, namely, $CRM\_forward(), cr\_div()$,

and $CRM\_backward()$, and one host function $calc\_dim()$ first we have to calculate the block size according to the size of matrix given by algorithm 2 and step number of forward and backward sub-steps for this we use one cycle

```
 for (i = 0;i < log2(n + 1) - 1;i++){
stepNum = (n - pow(2.0, i + 1)) / pow(2.0, i + 1) + 1;
calc_dim(stepNum, dimBlock, dimGrid);
CRM_forward <<<dimGrid, dimBlock >>>(d_a, d_b, d_c, d_f, n, stepNum, i)
  }
```

Algorithm2. Calculate block size

```
1.  Function calc_dim(int stepSize, dim3 *block, dim3 *grid) {
2. if (stepSize <4) { block->x = 1;block->y = 1; }
3. else if (stepSize <16) { block->x = 2;block->y = 2; }
4. else if (stepSize <64) { block->x = 4;block->y = 4; }
5. else if (stepSize <128) { block->x = 8;block->y = 8; }
6. else if (stepSize <256) { block->x = 16;block->y = 16; }
7. else if (stepSize <512) { block->x = 32;block->y = 32; }
8. else if (stepSize <1024) { block->x = 64;block->y = 64; }
9. else if (stepSize <2048) { block->x = 128;block->y = 128; }
10.      else if (stepSize <4196) { block->x = 256;block->y = 256; }
11.      grid->x = (unsigned int)ceil(sqrt((double) stepSize/ block->x));
12.      grid->y = (unsigned int)ceil(sqrt((double) stepSize/ block->y));
13.      End function
```

Here $log2(n+1)-1$ is step number and the variable $stepNum$ is for identify how much block size we need therefore the function $calc\_dim()$ identified block size after that the $CRM\_forward()$ function runs $log2(n+1)-1$ times consequently the system reduced one equation. After that we synchronize the device and will call the $cr\_div()$ function, this function calculate two unknowns, then we use again one cycle

```
for (i = log2(n + 1) - 2;i >= 0;i--) {
stepNum = (n - pow(2.0, i + 1)) / pow(2.0, i + 1) + 1;
calc_dim(stepNum, dimBlock, dimGrid);
CRM_backward<<<dimGrid, dimBlock >>>(d_a, d_b, d_c, d_f, d_x, n, stepNum,
    i);
}
```

Here backward substitution runs $log2(n+1)-2$ times because first backward substitution sub-step calculated by $calc\_dim()$ function thus we calculate d_x array

after that we will copy calculated data d_x from device to host using

```
cudaMemcpy(y, d_x, sizeof(double)*n, cudaMemcpyDeviceToHost)
```

2.3.4 Experimental results

Here we show the results obtained on a desktop computer with configuration 4352 cores GeForce RTX 2080 TI, NVIDIA GPU together with a CPU Intel Core(TM) i7-9800X, 3.80 GHz, RAM 64Gb. Simulation parameters are configured as follows. Mesh size is uniform in both directions with $\Delta x = \Delta y = 0.5$, and numerical time step Δt is 0.05 s, and simulation time is T=5.0 s, therefore the total number of time steps is 100. To present more realistic data, we tested six cases with domain sizes of 256×256,512×512,1024×1024,2048×2048,4096×4096,8192×8192.

In Table 2.1 we report the execution times in seconds for serial (CPU time) and CUDA (GPU time) implementation of cyclic reduction method to the problem (2.15)-(2.17).

Table 2.1 – Execution timing and speedup with the Intel Core(TM) i7-9800X, 3.80GHz, NVIDIA RTX 2080 TI

Domain sizes	CPU time	GPU time
256×256	0.74	0.52
512×512	3.39	3.01
1024×1024	15.46	9.5
2048×2048	65.95	23.6
4096×4096	318.42	79.66
8192×8192	2856.32	347.48

In this section, we have introduced a numerical solution of a two-dimensional tsunami wave equation based on an implicit finite difference scheme using the cyclic reduction method. We develop an approach parallelization of the cyclic reduction method on the graphic processing unit. And we showed how we accelerated the cyclic reduction method on the NVIDIA GPU. From the test results of table 2.3, it can be seen that the acceleration algorithm proposed by us gives a good result.

In future work, we would also like to use the OpenCL programming language instead of CUDA to make the code portable to non-NVIDIA hardware, including multicore systems. Other options to investigate could be the use of compiler directives, somewhat similar to the philosophy of OpenMP, or higher-level programming environments.

2.4 A PARALLEL HYBRID IMPLEMENTATION OF 2D ACOUSTIC WAVE EQUATION BASED ON IMPLICIT FINITE DIFFERENCE SCHEMES

In this section, we propose a hybrid parallel programming approach for a numerical solution of a two-dimensional acoustic wave equation using an implicit difference scheme for a single computer. The calculations were carried out in an implicit finite difference scheme. First, we will transform the differential equation into an implicit finite-difference equation and then using the ADI method we split the equation to 2 sub-equations. Using the cyclic reduction algorithm we will calculate an approximate solution. Then we will change this algorithm to parallelize this on GPU, GPU+Open MP, and Hybrid (GPU+Open MP+MPI). The special focus is on improving the performance of the parallel algorithms to calculate the acceleration based on the execution time. We show that the code that runs on the hybrid approach gives the expected results by comparing our results to those obtained by running the same simulation on a classical processor core, CUDA, and CUDA+Open MP implementation.

The reduction of computational time for long-term simulation of physical processes is a challenge and an important issue in the field of modern scientific computing. The cost of supercomputers, CPU clusters and hybrid clusters with a large number of GPUs are very expensive and they consume a lot of energy, which is inaccessible and ineffective to some small laboratories and individuals.

Nowadays, new generation computers are multi-core, hybrid architecture and their computational power is also quite high. For example, the Intel Xeon E5-2697 v2 (2S-E5) processors theoretically computing power about 19.56 GFLOPS, and, accordingly, the computational power of the NVIDIA TITAN Xp video card is about up to 379.7 GFLOPS. If we use the computing power of the CPU and GPU together, we can show good results.

The goal of this work is to develop a parallel hybrid implementation of the finite-difference method for solving two-dimensional wave equation using CUDA, CUDA + Open MP and CUDA + OpenMP + MPI technologies and studying the parallelization efficiency by comparing the time to solve this problem on the above various approaches. Already for several years, GPUs have been used to accelerate well parallelizable computing, only with the advent of a new generation of GPUs with multicore architecture, this direction began to give palpable results.

For multidimensional problems, the efficiency of an implicit compact difference scheme depends on the computational efficiency of the corresponding matrix solvers. From this point of view, the ADI method [37] is promising because they can decompose a multidimensional problem into a series of one-dimensional problems. It has been shown that schemes acquired are unconditionally stable. For the proper assignment of large domains of modeling, two- or three-dimensional computational grids with a sufficient number of points are used. Calculations on such grids require more CPU time

and computer memory resources. To accelerate the computation process, GPU, Open MP, MPI technologies were used in this paper, which allows the program to operate on larger grids. With GPU becoming a viable alternative to CPU for parallel computing, aforementioned parallel tridiagonal solvers and other hybrid methods have been implemented on GPUs [47]-[54]. In this paper, we propose 3 different parallel programming approaches using hybrid CUDA, Open MP and MPI programming for personal computers. There are many examples in the literature of successfully using hybrid approaches for different simulations [66]-[71]. Here we consider some issues in the numerical simulation of some problems in the propagation of waves in acoustic on high performance computing systems. We consider 2d acoustic wave equation in homogeneous medium

$$\frac{\partial^2 u}{\partial t^2} - c^2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = f(x,y,t), \big(t,(x,y)\big) \in [0;T] \times [0;l]; \quad (2.36)$$

subject to the initial conditions

$$u(0;\ x,y) = \ \varphi_1(x,y);\ x;\ y\ \in\ [0;\ l];\ \ (2.37)$$

$$\frac{\partial u(0,x,y)}{\partial t} = \varphi_2(x,y);\ x;\ y\ \in\ [0;\ l];\ \ (2.38)$$

and boundary conditions

$$u(t,x,0) = \ 0, u(t,x,l) = \ 0, t\ \in\ [0,T], x\ \in\ [0,l]; (2.39)$$

$$u(t,0,y) = \ 0, u(t,l,y) = \ 0, t\ \in\ [0,T], y\ \in\ [0,l]. (2.40)$$

We introduce a space-time grid with steps h_1, h_2, τ respectively, in the variables x, y, t:

$\omega_{h_1 h_2}^{\tau} = \{x_i = ih_1, i = \underline{0,N};\ y_j = jh_2,\ j = \underline{0,N};\ t_k = k\tau;\ k = \underline{0,M}, M\tau = T\}$ (2.41)

and on this grid we will approximate the differential problem (2.36) - (2.41) using the finite difference method. For problem (2.36) the ADI method has the form

$$\frac{u_{i,j}^{k+1/2} - 2u_{i,j}^{k} + u_{i,j}^{k-1/2}}{\tau^2} - \frac{c^2}{4h^2}\left(u_{i+1,j}^{k+\frac{1}{2}} - 2u_{i,j}^{k+\frac{1}{2}} + u_{i-1,j}^{k+\frac{1}{2}} + u_{i+1,j}^{k-\frac{1}{2}} - 2u_{i,j}^{k-\frac{1}{2}} + u_{i-1,j}^{k-\frac{1}{2}}\right) = f_{i,j}^{k} \quad (2.42)$$

$$\frac{u_{i,j}^{k+1} - 2u_{i,j}^{k+1/2} + u_{i,j}^{k}}{\tau^2} - \frac{c^2}{4}\left(u_{i,j+1}^{k+1} - 2u_{i,j}^{k+1} + u_{i,j-1}^{k+1} + u_{i,j+1}^{k} - 2u_{i,j}^{k} + u_{i,j-1}^{k}\right) = f_{i,j}^{k+\frac{1}{2}} \quad (2.43)$$

2.4.1 Hybrid parallel computing model

High-performance computing uses parallel computing to achieve high levels of performance. In parallel computing, the program is divided into many subroutines, and

then they are all executed in parallel to calculate the required values. In this section, we will propose a hybrid parallel approach numerical solving a two-dimensional wave equation, for this, we use CUDA, MPI, OpenMP technologies[82].

CUDA approach. The graphics processing unit (GPU) is a highly parallel, multi-threaded, and multi-core processor with enormous processing power. Its low cost and high bandwidth floating point operations and memory access bandwidth are attracting more and more high performance computing researchers [75]. In addition, compared to cluster systems, which consist of several processors, computing on a GPU is inexpensive and requires low power consumption with equivalent performance. In many disciplines of science and technology, users were able to increase productivity by several orders of magnitude using graphics processors [45,46]. The 2007, with the appearance of the CUDA programming language, programming GPUs on NVIDIA graphics cards became significantly simpler because its syntax is similar to C[62].

It is designed so that its constructions allow a natural expression of concurrency at the data level. A CUDA program consists of two parts: a sequential program running on the CPU, and a parallel part running on the GPU [46,64]. The parallel part is called the kernel. A CUDA program automatically uses more parallelism on GPUs that have more processor cores. A C program using CUDA extensions hand out a large number of copies of the kernel into available multiprocessors to be performed contemporaneously. The CUDA code consists of three computational steps: transferring data to the global GPU memory, running the CUDA core, and transferring the results from the GPU to the CPU memory. We have designed a CUDA program based on cyclic reduction method, whose the full CR function codes are located in [74]. The algorithm for solving the problem (1.1) is shown in Algorithm 1.

Algorithm 1 Implementation of 2D wave equation

compute initial condition matrix U0
from initial condition (1.2) we can get u=U0
while $(t < t_{end})$ **do**
  for j=0,...,n
  for i=0,...,n
  calculate tridiagonal system elements a_i, b_i, c_i, f_i
  call function $CR(a_i, b_i, c_i, f_i, y_i, n)$
  calculate matrix Ux
  for i=0,...,n
  for j=0,...,n
  calculate tridiagonal system elements aj, bj, cj, fj
  call function $CR(a_j, b_j, c_j, f_j, y_j, n)$
  calculate matrix Uy
  swap(u,Ux)
  swap(U0,Uy)
  $t \leftarrow t + \Delta x$

Here, u, U_0, U_x, U_y denote $u_{i,j}^{k-1/2}, u_{i,j}^{k}, u_{i,j}^{k+1/2}, u_{i,j}^{k+1}$ respectively.

The *CR()* function includes 3 device functions, namely, *CRM_forward(); cr div(),* and *CRM_backward(),* and one host function *calc_dim()* first we have to calculate the

block size according to the size of matrix and step number of forward and backward sub-steps for this we use one cycle

```
 for (i = 0;i < log2(n + 1) - 1;i++){
stepNum = (n - pow(2.0, i + 1)) / pow(2.0, i + 1) + 1;
calc_dim(stepNum, dimBlock, dimGrid);
CRM_forward <<<dimGrid, dimBlock >>>(d_a, d_b, d_c, d_f, n, stepNum, i)
  }
```

Here log2(n + 1) - 1 is step number and the variable stepNum is for identify how much block size we need therefore the function *calc_dim()* identified block size after that the *CRM_forward()* function runs log2(n+1)-1 times consequently the system reduced one equation. After that we synchronize the device and will call the *cr_div()* function, this function calculates two unknowns. Then we use again one cycle

```
for (i = log2(n + 1) - 2;i >= 0;i--) {
stepNum = (n - pow(2.0, i + 1)) / pow(2.0, i + 1) + 1;
calc_dim(stepNum, dimBlock, dimGrid);
CRM_backward<<<dimGrid, dimBlock >>>(d_a, d_b, d_c, d_f, d_x, n, stepNum,
    i);
}
```

here backward substitution runs log2(n + 1) - 2 times because first backward substitution sub-step calculated by calc dim() function thus we calculate d x array after that we will copy calculated data dx from device to host using cudaMemcpy(y, d_x; sizeof(double) *n, *cudaMemcpyDeviceToHost*)

OpenMP+CUDA approach. OpenMP (Open Multi-Processing) was introduced to provide the means to implement shared memory concurrency in FORTRAN and C/C ++ programs. In particular, OpenMP defines a set of environment variables, compiler directives and library procedures that will be used for parallelization with shared memory. OpenMP was specifically designed to use certain characteristics of shared memory architectures, such as the ability to directly access memory through-out a system with low latency and very fast shared memory locking [72]. We can easily to parallelize loops by using MPI thread libraries and involve the OpenMP compilers. In this way, the threads can obtain new tasks, the unprocessed loop iterations, directly from local shared memory. The basic idea behind OpenMP is data-shared parallel execution. It consists of a set of compiler directives, callable runtime library routines and environment variables that extend FORTRAN, C, and C++ programs.

The unit of workers in OpenMP threads. It works well, when accessing shared data costs you nothing. Every thread can access a variable in a shared cache or RAM. In this paper, we will use OpenMP to solve the initial condition of the problem. Because when solving the initial condition we use 2 cycles and we calculate one matrix for this,

OpenMP parallel computing model is very easy to implement parallelism, and it has low latency, high bandwidth.

Hybrid approach.

We use MPI technology to calculate the elements of the tridiagonal matrix system, i.e a_i; b_i; c_i; f_i because these values can be calculated independently, so we can successfully apply MPI technology here. Listing code 1 shows the program code.

LISTING 1. Calculation of ai, bi, ci, fi

```
 i1 = (n*rank) / size;
 i2 = (n*(rank + 1)) / size;
for (i = i1; i <i2; i++)
 {
  a_m[kk] = tau*tau;
  c_m[kk] = tau*tau;
  b_m[kk] = 2 * tau*tau + h*h;
  f_m[kk] = h*h*Unn[i] - 2 * h*h*uu0[i];
  kk++;
 }

 MPI_Gather(a_m, n/size, MPI_DOUBLE, a, n/size, MPI_DOUBLE, 0,
    MPI_COMM_WORLD);
 MPI_Gather(b_m, n/size, MPI_DOUBLE, b, n/size, MPI_DOUBLE, 0,
    MPI_COMM_WORLD);
 MPI_Gather(c_m, n/size, MPI_DOUBLE, c, n/size, MPI_DOUBLE, 0,
    MPI_COMM_WORLD);
 MPI_Gather(f_m, n/size, MPI_DOUBLE, f, n/size, MPI_DOUBLE, 0,
    MPI_COMM_WORLD);

 if (rank == 0)
 {
  MPI_Bcast(a, n, MPI_DOUBLE, 0, MPI_COMM_WORLD);
  MPI_Bcast(b, n, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Bcast(c, n, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Bcast(f, n, MPI_DOUBLE, 0, MPI_COMM_WORLD);
}
```

These parallel technologies, CODA, OpenMP and MPI can be combined to form a multilayered hybrid structure, the premise is that the system has several CPU cores and at least one graphics processor. Under this hybrid structure (Figure 2.12), we can make better use of the advantages of another programming model.

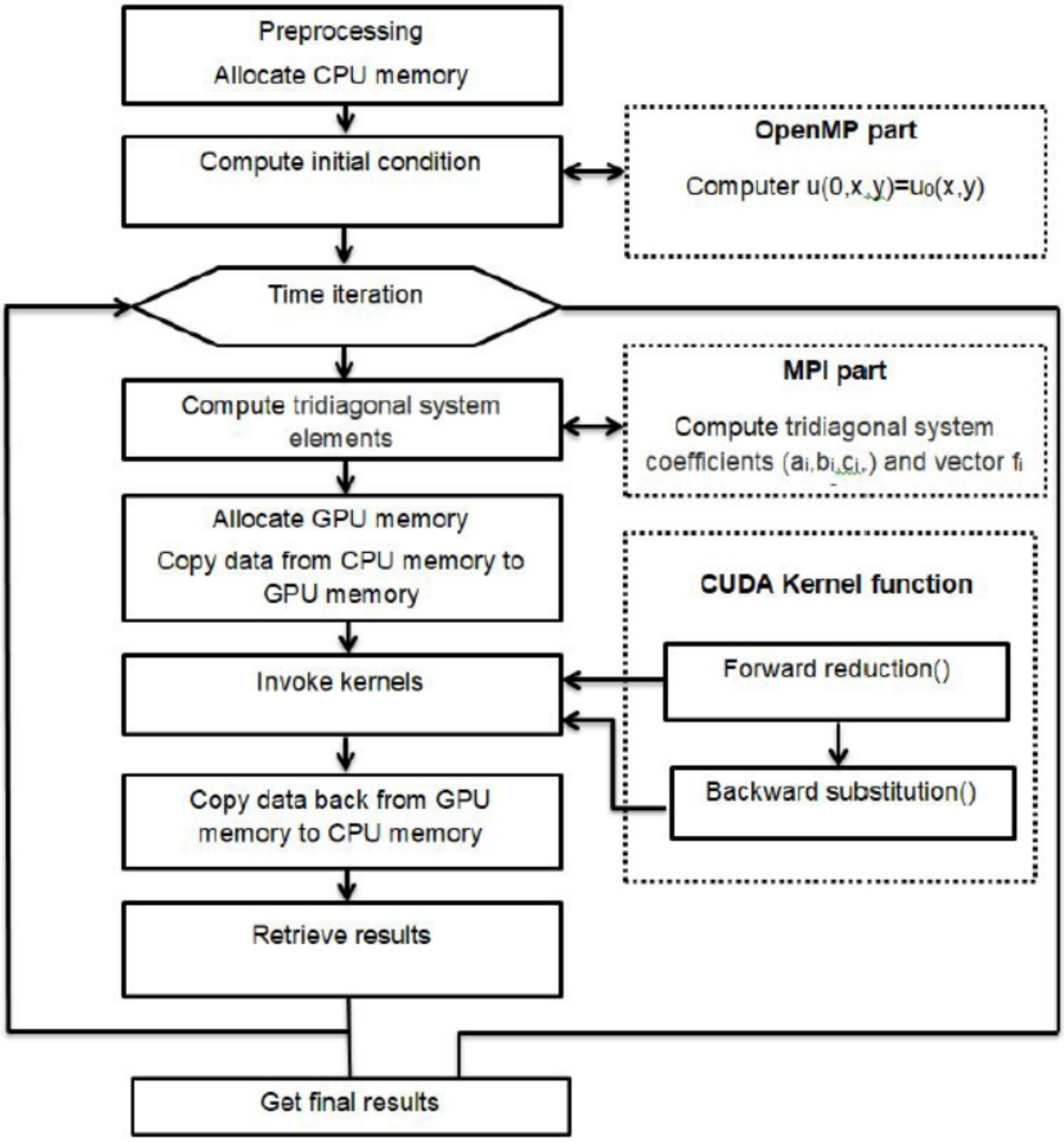


Figure 2.12 – Flowchart hybrid approach

2.4.2 Experimental results

In this section we show the results obtained on a desktop computer with configuration 4352 cores GeForce RTX 2080 TI, NVIDIA GPU together with a CPU Intel Core(TM) i7-9800X, 3.80 GHz, RAM 64Gb. Simulation parameters are configured as follows. Mesh size is uniform in both directions with $\Delta x = \Delta y = 0{,}01$, coefficients $c = 1$ and numerical time step Δt is 0.01, and simulation time is T=1.0, therefore the total numerical time step is 50.

Using the implicit subscheme (1.7), the cyclic reduction [61] method is performed in the x direction, with the result that we get the grid function $u_{i,j}^{k+1/2}$. In the second

fractional time step, using the subscheme (1.8), the Cyclic reduction method is performed in the direction of the y axis, with the result that we get the grid function k. The Cyclic reduction method has the order $O(\tau + h^2)$, i.e. the first order in time and the second in x and y variables.

To present more realistic data we tested four cases with large domain sizes of $1024 \times 1024, 2048 \times 2048, 4096 \times 4096,$ and 8192×8192.

In Table 2.2 we report the execution times in seconds for serial (CPU time), CUDA (GPU time), GPU+OpenMP, and CUDA+OpenMP+MPI implementation of cyclic reduction method to the problem (2.36) - (2.41).

Table 2.2 – Execution Time (Seconds)

N (mesh size)	CPU	GPU	GPU/Open MP	GPU/OpenMP/MPI		
				2 CPU core	4 CPU core	8 CPU core
1024× 1024	48.13	24.104	24.151	24.432	23.232	22.61
2048× 2048	189.677	45.033	45.01	35.133	33.571	30.261
4096× 4096	755.614	122.24	59.996	58.797	54.223	51.413
8192× 8192	3272.305	435.854	239.556	173.45	168.876	159.50

In this section, we proposed the numerical solution of a 2 acoustic wave equation based on an implicit finite difference scheme using the cyclic reduction method. And we have constructed a heterogeneous hybrid programming environment for a single PC by combining the message passing interface MPI, OpenMP, and CUDA programming. Also implemented parallelization of the cyclic reduction method on the graphic processing unit. And we showed how we accelerated the cyclic reduction method on the NVIDIA GPU. From the test results of table 1 it can be seen that the acceleration method proposed by us gives a good result. Our hybrid CUDA/OpenMP/CPU implementation obtained 10 -15% faster than CUDA implementation.

In future work, we plan to improve and adapt our hybrid approach for GPU clusters and test on a GPU cluster.

2.5 Conclusion

In the second chapter, we considered parallel numerical implementation hyperbolic type equations with singular coefficients. Firstly, we presented MPI implementation of 2D wave equation with a distributional coefficient then CUDA implementation of 2D tsunami wave equation and related computational results.

And, we presented a hybrid implementation of acoustic wave equation then we compared the results from the different implementations.

In a hybrid implementation, the joint use of Open MP, CUDA and MPI technologies to solve one problem, the result of the calculations showed that this implementation gives very good results.

3 SOFTWARE COMPLEX FOR THE INVESTIGATION OF THE WAVE PROPAGATION IN 1D AND 2D WAVE EQUATION WITH SINGULAR COEFFICIENTS

3.1Overview

In this chapter, we propose a software complex for numerical simulation of wave equation with singular coefficients therein the numerical method based on an implicit finite difference scheme which is unconditional stable. The application is written in Python, a modern object-oriented programming language, and it may run on all platforms. It is simple to use, the data is processed quickly and the results being presented in a plot and animation could save on disk. Currently, a lot of software is being developed for the numerical solution of wave equations, in particular, software that simulates tsunami waves, software that describes the pure wave equation, software that describes elastic waves, and more.

Oscillations and waves are extremely important phenomena in science. In nature, everywhere we can find oscillations and accurate estimation of wave propagation through, from the shaking of atoms to the large tsunami phenomena, is an important issue in science. Numerical simulation of wave propagation is fundamental in many scientific and engineering applications. An actual task today is the study of wave propagation in a discontinuous medium.

Wave equations are partial differential equations which describe the propagation of various types of waves, such as acoustic, elastic and electromagnetic waves[76]. Many mathematical studies have been done on the wave equation of the singular coefficient [p4,6,7,18,8,33,32,43].

The numerical solution of the wave equation began a long time ago and is developing to this day, there is a large amount of work devoted to numerical methods developed for the study of wave processes in recent decades. It includes a finite-difference method [25], a finite-volume method [26], the finite-element method [78], a spectral-element method [79] a two-level compact ADI method [29] , the implicit Finite Difference Time Domain Methods [21], a boundary [integral] element method [27], and spectral methods [36]. Most models have been developed for technical applications.

In the last decade, many numerical modeling software have been developed, most of them are designed to model hydrodynamic processes, chemical reactions, and tsunami wave propagation.

In this chapter, we describe a software complex for numerical simulation of the wave equation with singular coefficients. It is based on highly accurate numerical methods on Cartesian grids. The computational domain is approximated with a Cartesian grid where high order fully implicit finite differences schemes are easily implemented and very efficient and part of program code are given [84].

3.2 Mathematical model

We describe the one-dimensional and two-dimensional wave equations with time-varying coefficients that characterize the propagation of waves in a continuous medium. These equations are very important in the fields of engineering and physics, mathematics, and many scientists are working on these equations.

1D model

$$\frac{\partial^2 u(t,x)}{\partial t^2} - a(t)\frac{\partial^2 u(t,x)}{\partial x^2} + b(t)\frac{\partial u(t,x)}{\partial t} + c(t)\frac{\partial u(t,x)}{\partial x} + d(t)u(t,x) = f(t,x), \quad (t,x) \in (0,T) \times \Omega \tag{3.1}$$

The initial values are given by the initial value condition

$$u(0,x) = \varphi(x), \quad \frac{\partial u}{\partial t}(0,x) = \psi(x), \quad (x) \in \underline{\Omega}, \tag{3.2}$$

where $\underline{\Omega} = (0,X)$. Since the domain Ω is bounded we require some conditions on its boundary, for example, the Dirichlet boundary condition

$$u(t,x)|_{\partial\Omega} = g(t,x), \quad t \in [0,T]. \tag{3.3}$$

where a(t),b(t),c(t),d(t) are $\delta-$ like singularity function.

2D model

$$\frac{\partial^2 u(t,x,y)}{\partial t^2} - a(t)\left(\frac{\partial^2 u(t,x,y)}{\partial x^2} + \frac{\partial^2 u(t,x,y)}{\partial y^2}\right) + b(t)\frac{\partial u(t,x,y)}{\partial t} + c(t)\left(\frac{\partial u(t,x,y)}{\partial x} + \frac{\partial u(t,x,y)}{\partial y}\right) + d(t)u(t,x,y) = f(t,x,y), \quad (t,x,y) \in (0,T) \times \Omega \tag{3.4}$$

where $\Omega = (0,X) \times (0,Y)$ for some fixed $X, Y > 0$. The initial values are given by the initial value condition

$$u(0,x,y) = \varphi(x,y), \quad \frac{\partial u}{\partial t}(0,x,y) = \psi(x,y), \quad (x,y) \in \underline{\Omega}, \tag{3.5}$$

where $\underline{\Omega} = (0,X) \times (0,Y)$. Since the domain Ω is bounded we require some conditions on its boundary, for example, the Dirichlet boundary condition

$$u(t,x,y)|_{\partial\Omega} = g(t,x,y), \quad t \in [0,T]. \tag{3.6}$$

2.3.1 Finite difference scheme and calculation algorithm

In this section, we consider the 2D equation, therein include 1D equation.

We introduce space-time grids with steps τ, h_1, h_2 in the variables t, x, y, respectively, that are

$$\omega^{\tau}_{h_1,h_2} = \{(t_k, x_i, y_j): t_k = k\tau; x_i = ih_1; y_j = jh_2, \quad (k,i,j) \in I\}, \Omega^{\tau}_{h_1,h_2} = \{(t_k, x_i, y_j): t_k = k\tau; x_i = ih_1; y_j = jh_2, \quad (k,i,j) \in \underline{I}\}, \tag{3.7}$$

where

$$I := \{(k,i,j) \in Z^3_+: 0 < k \le M; 0 < i < N_1; 0 < j < N_2\},$$

$$\underline{I} := \{(k,i,j) \in Z^3_+: 0 \le k \le M; 0 \le i \le N_1; 0 \le j \le N_2\},$$

and

$$X = h_1 N_1, Y = h_2 N_2, T = \tau M.$$

One calculates an approximate solution from discrete points in the time-spatial grid $\Omega^{\tau}_{h_1,h_2}$. On this grid we approximate the problem (3.4)–(3.6) using the finite difference method. For simplicity, we put $N := N_1 = N_2$ and denote $h := h_1 = h_2$. Consider the Crank-Nicolson scheme for the problem (3.4)–(3.6). First we transform the partial differential equation (3.4) into the implicit finite-difference equation

$$\frac{u_{i,j}^{k+1}-2u_{i,j}^{k}+u_{i,j}^{k-1}}{\tau^2} - \frac{a^k}{4h^2}((u_{i+1,j}^{k+1} - 2u_{i,j}^{k+1} + u_{i-1,j}^{k+1} + u_{i+1,j}^{k-1} - 2u_{i,j}^{k-1} + u_{i-1,j}^{k-1}) + (u_{i,j+1}^{k+1} - 2u_{i,j}^{k+1} + u_{i,j-1}^{k+1} + u_{i,j+1}^{k-1} - 2u_{i,j}^{k-1} + u_{i,j-1}^{k-1})) + b^k(\frac{u_{i,j}^{k+1}-u_{i,j}^{k}}{\tau}) + c^k(\frac{u_{i+1,j}^{k}-u_{i,j}^{k}}{h}) + d^k u_{i,j}^{k} = f_{i,j}^{k} \quad (3.8)$$

for $(k,i,j) \in \omega^{\tau}_{h_1,h_2}$, where $a^k := a(k\tau), b^k := b(k\tau), c^k := c(k\tau), d^k := d(k\tau)$, $f_{i,j}^{k} := f(k\tau, ih, jh)$ with initial conditions

$$u_{i,j}^{0} = \varphi_{i,j}, \;\; u_{i,j}^{1} - u_{i,j}^{0} = \tau\varphi_{i,j}, \quad (3.9)$$

for $(i,j) \in \underline{0,N} \times \underline{0,N}$, and with boundary conditions

$$u_{0,j}^{k} = 0, \;\; u_{N,j}^{k} = 0, \;\; u_{i,0}^{k} = 0, \;\; u_{i,N}^{k} = 0, \quad (3.10)$$

for $(j,k) \in \underline{0,N} \times \underline{0,M}$ and $(i,k) \in \underline{0,N} \times \underline{0,M}$, respectively.

It is well-known, that the implicit scheme (3.8)–(3.10) is unconditionally stable and it has accuracy order $O(\tau + |h|^2)$, see, for example [16]. We solve the difference equation (3.8) by the alternating direction implicit (ADI)method, namely, dividing it into two sub problems [29]

For simulation we use initial condition:

$U(x,y,0) = \begin{cases} (x-0.4)(x-0.6)+(y-0.4)(y-0.6), & \text{if} 0.4 < x < 0.6 \text{and} 0.4 < y < 0.6; \\ 0, & \text{else} \end{cases}$

$U_t(x,y,0) = 0$ and dirichlet boundary conditions.

3.3 Software package description

The software we offer is an open-source and high-precision calculation tool, where the algorithm is based on an implicit difference scheme and is an absolute stable. The calculations include the following steps:

1. Selection of coefficients of variables depending on the given equation
2. Setting calculation parameters, ie setting area size, spatial and temporal steps, and other parameters
3. Carrying out calculations, viewing the solution of the problem in the form of animation
4. Compare the obtained results and save it in files, if necessary

The software was developed in Python 3.8 using the matplotlib libraries (graphing), tkinter (creating a graphical interface)

Table 3.1– Software structure

module	function
$one\_dim\_solver.py$	one-dimensional equation solver
$two\_dim\_solver.py$	two-dimensional equation solver
$gui.py$	Building a graphical user interface
$plots.py$	Graphing the results of the equation
$main.py$	The program launch module, which calls the remaining modules and provides data exchange between them

3.3.1 Graphical user interface

Figure 3.1 shows the graphical interface of the software package. In the left half - the control panel consists of the following blocks: “choose singular coefficient” - choose one of the coefficients a (t), b (t), c (t), d (t) depending on the equation; “Set domain size” - Set out of the computational area, the calculation time and steps of the calculation grids; “Set parameters” - assign the location of the initial wave and other additional parameters; “Set exponential parameters” - assign the degree of the selected singular coefficient; “Status screen” - a window showing the calculation process; "Control Panel" - start and stop the computational experiment.

In the right half is an equation and a window for graphically displaying the calculation results.

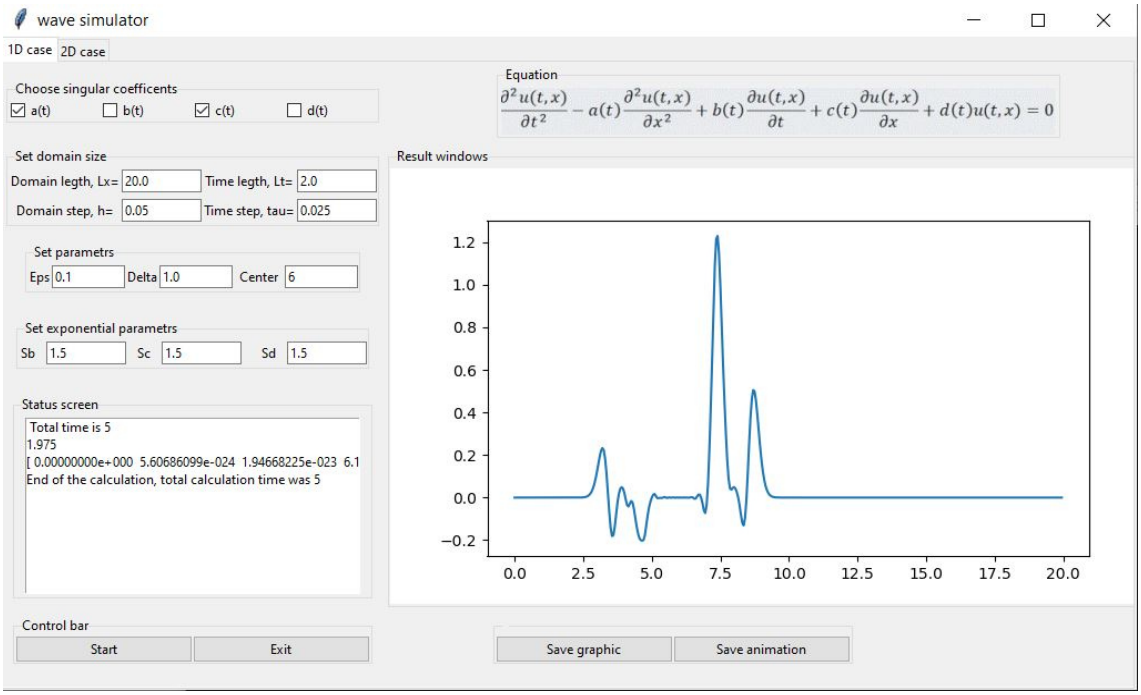


Figure 3.1 – Graphical user interface for 1D case

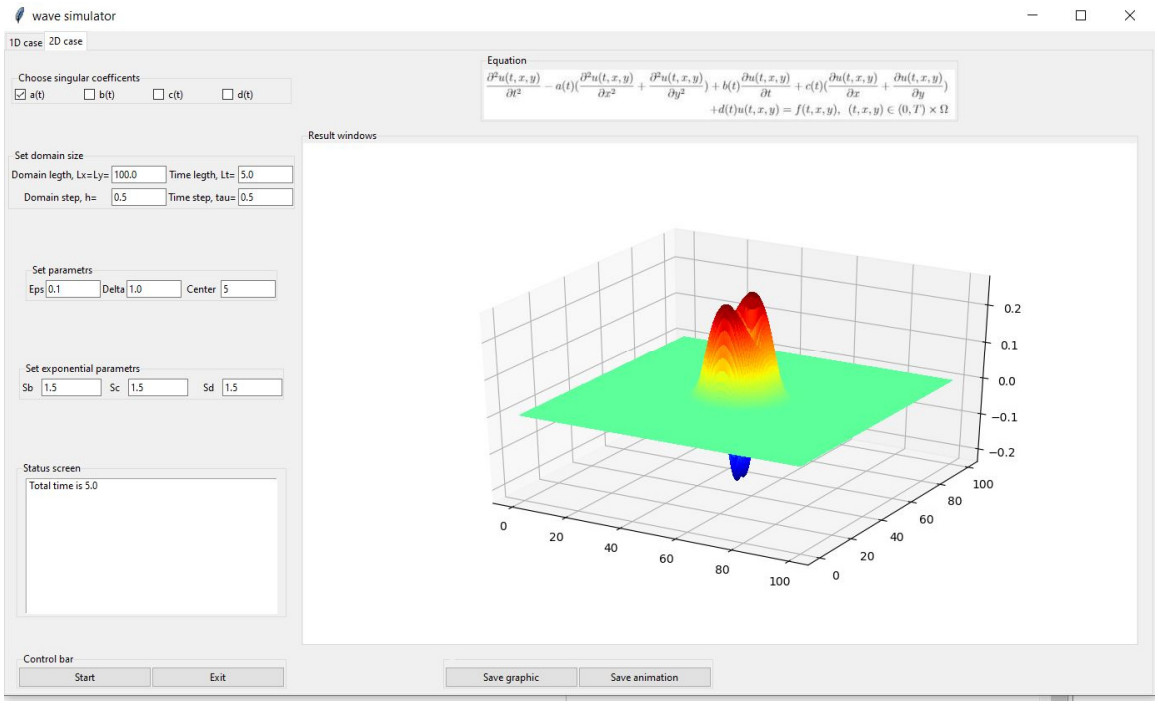
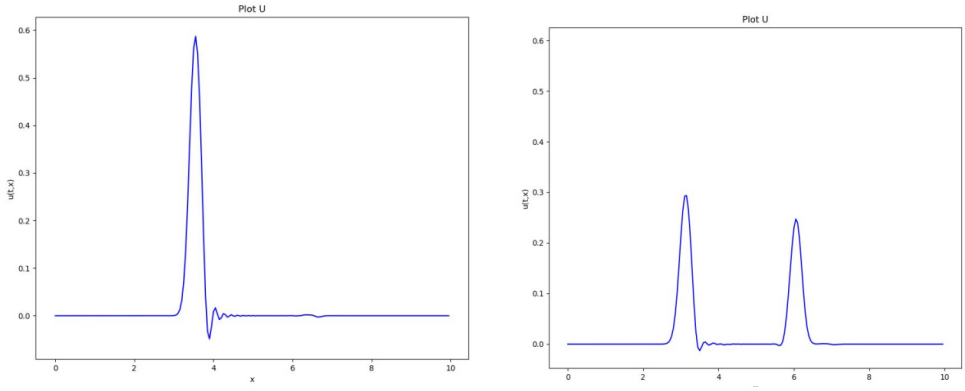


Figure 3.2 – Graphical user interface for 2D case

3.4 Calculation example

For the computational experiment, the following parameters were selected: for the one-dimensional case, the domain length Lx = 20.0, the grid step h = 0.5, the simulation time Lt = 2.0, the time step tau = 0.05, the locations with the initial wave center = 6.0, the degree for all of the coefficient is 1.5; for the two-dimensional case, the domain length is Lx = Ly = 100.0, the grid step is h = 0.5, the simulation time is Lt = 5.0, the time step is tau = 0.05, the initial wave position is center = 5.0, and the degree for all of the coefficient is 1.5.



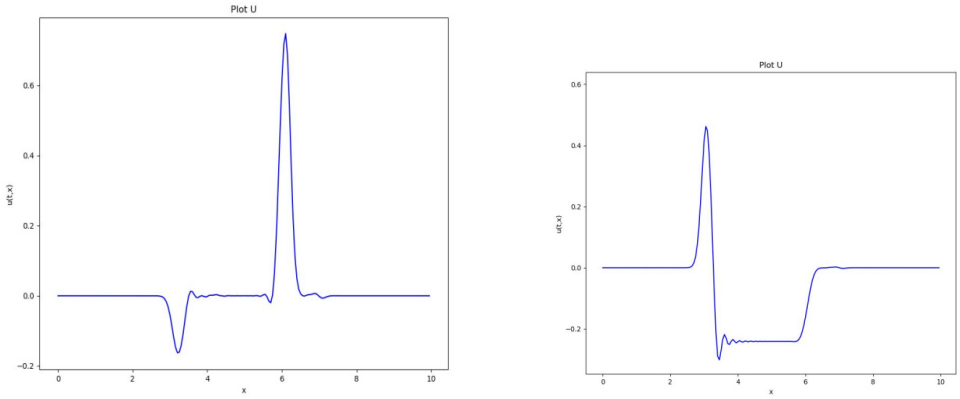


Figure 3.3 – 1D simulation results regarding coefficients a(t),b(t),c(t),d(t)

Figure 3.3 shows the results after a separate choice of the coefficients a (t), b (t), c (t), d (t). In all 4 cases, the main calculation parameters are the same, and from these figures it is clear that when the coefficient a(t) is selected, the wave does not change, when the coefficient b(t) is selected, half the wave returns, when the coefficient c(t) is selected, the wave returns completely, when the coefficient d (t) is selected, a very interesting situation arises when the wave reaches a critical time, the wave height does not change and the lower part descends, the descending parts of the wave come back.

Developed software for conducting a computational experiment provides calculation and modeling for studying the propagation of waves in which the computation algorithm based on an absolute stable fully implicit finite difference scheme. The computation module provides for the possibility of saving the whole simulation in the format animation and in pictures. The obtained results might be useful for studying wave propagation through a discontinuous medium.

CONCLUSION

In this thesis, we have considered the numerical solution of hyperbolic type equations with singular coefficients. As a result of comparing explicit and implicit finite difference schemes, we have chosen an implicit finite difference scheme that is absolutely stable. Sequential algorithm for numerical solution, implemented by the Thomas method.

We used an implicit difference scheme to numerically solve the tsunami and acoustic wave equations. We have theoretically proved that there exists a very weak solution of the tsunami equation with a singular coefficient. When solving the Tsunami equation, we investigated its coefficient under singular and non-singular cases. We used this tsunami model to simulate a tsunami in the Caspian Sea and made different predictions depending on the height of the initial wave.

Due to the fact that the computation time for the sequential algorithm was very long in the long-term modeling of a large area, therefore, we began to parallelize the sequential algorithm.

When parallelizing the Thomas method in the MPI environment, we used the parallel method proposed by Yanenko. We have developed a parallel algorithm using the Yanenko method for the numerical solution of a two-dimensional wave equation with a singular coefficient in the MPI environment, the results are tested on a personal computer and on a computing cluster. The test results show that the method of Yanenko is much more effective on individual nodes than several nodes. This means that this method is more effective on personal computers than a cluster because it takes a lot of time to exchange data between nodes.

We have developed a parallel algorithm for solving a two-dimensional tsunami equation using CUDA technology. The parallel algorithm is based on the cyclic reduction method. The test results showed that this parallel algorithm gives good results.

We have developed a parallel hybrid algorithm for the numerical solution of a two-dimensional acoustic wave equation for a single PC by combining the message passing interface MPI, Open MP, and CUDA programming. Our hybrid CUDA/Open MP/CPU implementation obtained 10 -15% faster than CUDA implementation.

For the convenience of studying one-dimensional and two-dimensional wave equation with singular coefficients, we have developed open-source cross-platform software. The computation algorithm of the software based on an absolute stable fully implicit finite difference scheme. This software gives the possibility to view the animation of the equation and save the simulation results of each time step as a picture.

REFERENCES

1 Imamura F., Yalcine A.C. Tsunami Modeling Manual. – 2006. – P. 1-58.

2 Jaiswal R. K., Rastoqi B. K., Murty T. S. Tsunamigenic Sources in The Indian Ocean // Science of Tsunami Hazards. – 2008. – Vol. 27. № 2. – P. 32-48

3 Knut-Andreas Lie. The Wave Equation in 1D and 2D. https://www.uio.no/studier/emner/matnat/ifi/nedlagteemner/INF2340/v05/foiler/sim04.pdf

4 Garetto C., Ruzhansky M. Hyperbolic Second Order Equations with Non-Regular Time Dependent Coefficients // Archive for Rational Mechanics and Analysis. – 2015. – Vol. 217. №1. – P. 113–154.

5 Garetto C., Ruzhansky M. Hyperbolic second order equations with non-regular time de-pendent coefficients // Archive for Rational Mechanics and Analysis. – 2015. – Vol 217. №1. – P. 113–154.

6 Ruzhansky M., Tokmagambetov N. Very weak solutions of wave equation for Landau Hamiltonian with irregular electromagnetic field // Letters in Mathematical Physics. – 2017. – Vol. 107. №4. – P. 591–618.

7 Ruzhansky M., Tokmagambetov N. Wave equation for operators with discrete spectrum and irregular propagation speed // Archive for Rational Mechanics and Analysis. – 2017. – Vol. 226. № 3. – P. 1161–1207.

8 Munoz J.C., Ruzhansky M., Tokmagambetov N. Wave propagation with irregulardissipation and applications to acoustic problems and shallow water // Journal de Mathématiques Pures et Appliquées. – 2019. – Vol. 123. – P. 127–147.

9 Sebih M.E., Wirth J. On a wave equation with singular dissipation //Preprint, Arxiv:2002.00825 (2020)

10 Garetto C. On the wave equation with multiplicities and space-dependent irregular coefficients // Preprint, arXiv:2004.09657 (2020).

11 Altybay A., Ruzhansky M., Sebih M. E., Tokmagambetov N. Fractional Klein-Gordon equation with strongly singular mass term // Chaos, Solitons & Fractals. – 2021. –Vol. 143. – P. 110579-110647

12 Altybay A., Ruzhansky M., Sebih M. E., Tokmagambetov N. Fractional Schrödinger Equations with potentials of higher-order singularities // Reports on Mathematical Physics. – 2021. – Vol. 87. №1. – P. 129-144.

13 Altybay A., Ruzhansky M., Sebih M. E., Tokmagambetov N. The heat equation with singular potentials // Applied Mathematics and Computation. – 2021. – Vol. 399. – P. 126-132.

14 Friedlander F.G., Joshi M. Introduction to the Theory of Distributions. Cambridge University Press. – 1998.

15 Ebert M. R., Reissig M. Methods for Partial Differential Equations. Birkhäuser, – 2018.

16 Samarskii A.A. The Theory of Difference Schemes. CRC Press, – 2001.

17 National Oceanic and Atmospheric Administration. https://maps.ngdc.noaa.gov/viewers/bathymetry/

18 Ruzhansky M., Tokmagambetov N. On a very weak solution of the wave equation for a Hamiltonian in a singular electromagnetic field // Mathematical Notes. – 2018. – Vol.103. №5. – P. 856-858.

19 Cohen G. High-Order Numerical Methods for Transient Wave Equations, Springer, NewYork, – 2002. – P. 85-88.

20 Yanenko N.N. The Method of Fractional Steps, Springer-Verlag Berlin Heidelberg, – 1971. – P. 75-79.

21 Rouf H.K. 1971, Implicit Finite Difference Time Domain Methods. Theory and Applications/Hasan Khaled Rouf. – LAP Lambert Academic. – 2011. – P. 62-68.

22 Causon D.M., Mingham C.G., Qian L. Developments In Multi-Fluid Finite Volume Free Surface Capturing Methods // Advances in Numerical Simulation of Nonlinear Water Waves. – 2010. – Vol. 11. – P. 397-427.

23 Dehghan M., Mohebbi A. The combination of collocation, finite difference, and multigrid methods for solution of the two-dimensional wave equation // Numerical Methods Partial Differential Equation. – 2008. – Vol. 24. – P. 897-910.

24 Ding H.F., Zhang Y.X. A new fourth-order compact finite difference scheme for the two dimensional second-order hyperbolic equation // Journal of Computational and Applied Mathematics. – 2009. – Vol. 72, – P. 626-632.

25 Engsig-Karup A.P., Harry B., Bingham H.B., Lindberg O. An efficient flexible-order model for 3D nonlinear water waves // Journal of Computational Physics. – 2009. – Vol. 228. №6. – P. 2100-2118.

26 Greaves D. Application Of The Finite Volume Method To The Simulation Of Nonlinear Water Waves // Advances in Numerical Simulation of Nonlinear Water Waves. – 2010. – Vol.11. – P. 357-396.

27 Grue J., Fructus D. Model For Fully Nonlinear Ocean Wave Simulations Derived Using Fourier Inversion Of Integral Equations In 3D // Advances in Numerical Simulation of Nonlinear Water Waves. – 2010. – Vol. 11. – P. 1-42.

28 Kreiss H.O., Petersson N.A., Ystrom J. Difference approximation for the second-order wave equation // SIAM Journal on Numerical Analysis. – 2002. – Vol. 40. – P. 1940-1967.

29 Liao H. L. and Sun Z.Z. A two-level compact ADI method for solving second-order wave equations // International Journal of Computer Mathematics. – 2013. – Vol. 90. №7. – P. 1471-1488.

30 Liu J.M., Tang K.M. A new unconditionally stable ADI compact scheme for the two space-dimensional linear hyperbolic equation // International Journal of Computer Mathematics. – 2010. – Vol. 87. №10. – P. 2259-2267.

31 Munoz J.C., Ruzhansky M., Tokmagambetov N. Wave propagation with irregular dissipation and applications to acoustic problems and shallow waters // Journal de Mathématiques Pures et Appliquées. – 2019. – Vol. 123. – P. 127-147.

32 Munoz J.C., Ruzhansky M., Tokmagambetov N. Acoustic and shallow water wave propagations with irregular dissipation // Functional Analysis and Its Applications. – 2019. – Vol. 53. – P. 153-156.

33 Ruzhansky M., Tokmagambetov N. Wave Equation for 2D Landau Hamiltonian//Applied and Computational Mathematics. – 2019. – Vol. 18. №1. – P. 69-78.

34 Sapronov I. S., Bykov A.N. Parallel'no-konveyernyy algoritm [Pipelined Thomas algorithm] // Atom. – 2009. – Vol. 44. – P. 24-25.

35 Yanenko N.N., Konovalov A.N., Bugrov A.N., Shustov G.V. Ob organizatsii parallel'nykh vychisleni i "rasparallelivanii" progonki [On organization of parallel computations and parallelization of the tridiagonal matrix algorithm], // Numerical Methods of Continuum Mechanics. – 1978. – Vol. 9. №7. – P. 139-146.

36 Ducrozet G., Bonnefoy F., Touze D.L., Ferrant P. Open-source solver for nonlinear waves in open ocean based on High-Order Spectral method // Computer Physics Communications. – 2016. – Vol. 203. – P. 245-254.

37 Peaceman D. W., Rachford H. H. The Numerical Solution of Parabolic and Elliptic Differential Equations // Journal of the Society for Industrial and Applied Mathematics. – 1955. – Vol. 3. №1. – P. 28-41.

38 Craig I.J.D., Sneyd A.D. An alternating-direction implicit scheme for parabolic equations with mixed derivatives // Computers and Mathematics with Applications. – 1988. – Vol. 16. №4. – P. 341-350.

39 Douglas J., Rachford H.H. On the numerical solution of heat conduction problems in two and three space variables // Transaction of the American Mathematical Society. – 1956. – Vol. 82. – P. 421-489.

40 Douglas J. J., James E. G. A general formulation of alternating direction methods // Numerische Mathematik. – 1964. – Vol. 6. №1.– P. 428 453.

41 Jiao Y.Y., Zhao Q., Wang L., Huang G.H., Tan F. A hybrid MPI-OpenMP parallel computing model for spherical discontinuous deformation analysis // Computers and Geotechnics. – 2019. – Vol. 106. – P. 217-227.

42 Fedorov A.A., Bykov A.N., The method of two-level parallelization for solving tridiagonal systems on hybrid computers with multicore coprocessors [In Russian] // Parallel computational technologies. – 2016. – Vol. 17. – P. 234-244.

43 Altybay A., Ruzhansky M., Tokmagambetov N. Wave equation with distributional propagation speed and mass term: numerical simulations // Applied Mathematics E-Notes – 2019. – Vol. 19. – P. 552-562.

44 Klockner A., Warburton T., Bridge J., Hesthaven J.S. Nodal discontinuous Galerkin methods on graphics processors // Journal of Computational Physics – 2009. – Vol. 228. №21. – P. 7863-7882.

45 Bell N., Garland M. Efficient sparse matrix-vector multiplication on CUDA, NVIDIA Technical Report, – 2008. – P. 73-78.

46 Elsen E., LeGresley P., Darve E. Large calculation of the flow over a hypersonic vehicle using a GPU // Journal of Computational Physics. – 2008. – Vol. 227. – P. 10148-10161.

47 Zhang Y., Cohen J., Owens J. Fast tridiagonal solvers on the GPU // ACM Sigplan Notices. – 2010. – Vol. 45. №5. – P.127-136.

48 Zhang Y., Cohen J., Davidson A., Owens J. A hybrid method for solving tridiagonal systems on the GPU // GPU Computing Gems Jade Edition. – 2011. – Vol.117. – P. 22-32.

49 Davidson A., Owens J. Register packing for cyclic reduction: a case study // Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units, ACM. – 2011. – P. 4–11

50 Davidson A., Zhang Y., Owens J. An auto-tuned method for solving large tridiagonal systems on the GPU // Parallel and Distributed Processing Symposium (IPDPS), IEEE International, IEEE, – 2011. – P. 956-965.

51 Goddeke D., Strzodka R. Cyclic reduction tridiagonal solvers on GPUs applied to mixed-precision multigrid. Parallel and Distributed Systems, // IEEE Transactions. 2011. – Vol. 22. №1. – P. 22-32.

52 Kim H., Wu S., Chang L., Hwu W. A scalable tridiagonal solver for GPUs // Parallel Processing (ICPP), 2011 International Conference on, IEEE. 2011. – P. 444-453.

53 Sakharnykh N. Tridiagonal solvers on the GPU and applications to fluid simulation // GPU Technology Conference, – 2009. – P. 49–57

54 Wei Z., Jang B., Zhang Y., Jia Y. Parallelizing Alternating Direction Implicit Solver on GPUs // International Conference on Computational Science, ICCS, Procedia Computer Science. – 2013. – Vol. 18. – P. 389-398.

55 Diaz M. A., Solovchuk M., Tony Sheu W.H. High-Performance MultiGPU Solver for Describing Nonlinear Acoustic Waves in Homogeneous Thermoviscous Media // Computers and Fluids. – 2018. – Vol. 173. – P. 195-205.

56 Michea D., Komatitsch D. Accelerating a three-dimensional finite-difference wave propagation code using GPU graphics cards // Geophysical Journal International. – 2010. – Vol.182. – P. 389-402.

57 Mehra R., Raghuvanshi N., Savioja L., Lin M., Manocha D. An efficient GPU-based time domain solver for the acoustic wave equation // Applied Acoustics. – 2012. – Vol.73. – P. 83-94.

58 Lastra M., Castro M.J., Ureñaa C., Asunción M. Efficient multilayer shallow-water simulation system based on GPUs // Mathematics and Computers in Simulation. – 2018. – Vol. 148. – P. 48-65.

59 Lacasta A., Hernandez M.M., Murillo J., Navarro P.G. GPU implementation of the 2D shallow water equations for the simulation of rainfall/runoff events // Environmental Earth Sciences. – 2015. – Vol. 74. – P. 7295–7305.

60 Weiss R.M., Shragge J. Solving 3D anisotropic elastic wave equations on parallel GPU devices // Geophysics. – 2013. – Vol. 78. №2. – P. 1-9.

61 Hockney R.W. A fast direct solution of Poisson's equation using Fourier analysis. // Journal of the ACM. – 1965. – Vol. 12. №1. – P. 95-113.

62 NVIDIA, Nvidia, http://www.nvidia.com/. – 2019. – P. 150-152.

63 Song P., Zhang Z., Liang L., Zhang Q., Zhao Q. Implementation and performance analysis of the massively parallel method of characteristics based on GPU // Annals of Nuclear Energy. – 2019. – Vol. 131. – P. 257-272.

64 Nickolls J.I., Buck M., Garland K. Skadron. Scalable parallel programming with cuda // Queue. – 2008. – Vol. 6. №2. – P. 40-53.

65 NVIDIA TURING GPU ARCHITECTURE https://www.nvidia.com/content/dam/en-zz/Solutions/design visualization /turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf

66 Bodin F., Bihan S. Heterogeneous multicore parallel programming for graphics processing units //Scientific Programming – 2009. – Vol. 17. №4. – P. 325-336.

67 Yang C.T., Huang C. L., Lin C. F. Hybrid CUDA, OpenMP, and MPI parallel programming on multicore GPU clusters // Computer Physics Communications. – 2011. – Vol. 182, – P. 266-269.

68 Liu Y., Xiong R. A MPI + OpenMP + CUDA Hybrid Parallel Scheme for MT Oc-cam Inversion // International Journal of Grid and Distributed Computing. – 2016. – Vol. 9. №9. – P. 67-82.

69 Lamas Daviña A., Roman J.E. MPI-CUDA parallel linear solvers for block-tridiagonal matrices in the context of SLEPcs eigensolvers, Parallel Computing. – 2018. – Vol. 74. – P. 118-135.

70 Mu D., Chen P., Wang L. Accelerating the discontinuous Galerkin method for seismic wave propagation simulations using multiple GPUs with CUDA and MPI // Earthquake Science. – 2013. – Vol. 26. №6. – P. 377-393.

71 Alonso P., Cortina R., Martnez-Zaldvar F.J., Ranilla J. Neville elimination on multi- and many- core systems: OpenMP, MPI and CUDA // The Journal of Supercomputing. – 2011. – Vol. 58. – P. 215-225.

72 Karniadakis G., Kirby R.M. Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and Their Implementation Cambridge University Press, PA-P/CDR edition. – 2003. – P. 17-30.

73 Goddeke D., Strzodka R., Mohd-Yusof J., McCormick P., Buijssen S., Grajewski M., Tureka S. Exploring weak scalability for FEM calculations on a GPU-enhanced cluster // Parallel Computing. – 2007. – Vol. 33. – P. 685-699.

74 2D wave GPU implementation, https://github.com/Arshynbek/2Dwave-GPU-implementation

75 Klockner A., Warburton T., Bridge J., and Hesthaven J.S. Nodal discontinuous Galerkin methods on graphics processors // Journal of Computational Physics. – 2009. – Vol. 228. №21. – P. 7863-7882.

76 Mönkölä, Sanna, Numerical simulation of fluid-structure interaction between acoustic and elastic waves, University of Jyväskylä. – 2011. – P. 165-180.

77 Olenkov V.D., Lazareva I.V. and Biryukov A. D. Numerical simulation of wind flow around building complex with different software approaches // OP Conference Series: Materials Science and Engineering. – 2019. – Vol. 687. №5.

78 Richter G. An explicit finite element method for the wave equation // Applied Numerical Mathematics. – 1994. – Vol. 16. – P. 65-80.

79 Komatitsch D., Tromp J. Spectral-element simulations of global seismic wave propagation: II — Three-dimensional models, oceans, rotation and self-gravitation Geophysical Journal International. – 2002. – Vol. 150. №1. – P. 303-318.

80 Schwartz, L.: Sur l'impossibilité de la multiplication des distributions *// C. R.* Acad. Sci. Paris. – 1954. – Vol. 239. – P. 847-848,

81 Altybay A., Ruzhansky M., Sebih M. E., Tokmagambetov N. Tsunami propagation for singular topographies // Preprint, arXiv:2005.11931 (2020).

82 Altybay A., Ruzhansky M., Tokmagambetov N. A parallel hybrid implementation of the 2D acoustic wave equation // International Journal of Nonlinear Sciences and Numerical Simulation. – 2020. – Vol. 21, Iss. 7-8. – P. 821-827.

83 Altybay A., Tokmagambetov N. On numerical simulations of the 1D wave equation with a distributional coefficient and sourse term // International Journal of Mathematics and Physics. Al-Farabi Kazakh National University. – 2017. – Vol. 8. №2. – P. 28-33.

84 Altybay A., Tokmagambetov N. A parallel algorithm for solving the two-dimensional wave equation with a singular coefficient // KazNTU Bulletin. – 2019. – Vol. 1. – P. 404-410.

85 Altybay A., Tokmagambetov N. MPI parallel implement of a wave equation using an implicit finite difference scheme. // KBTU Bulletin. – 2020. №1(52). – P. 112-120.

86 Altybay A., Tokmagambetov N. GPU computing for 2d wave equation based on implicit finite difference schemes // Bulletin NIA RK. – 2020. №3(77). – P. 32-42.

87 Altybay A. Numerical simulation of one hyperbolic type equation with a delta-like coefficient // International Scientific Conference of Students and Young Scientists «Farabi alemi», Almaty, Kazakhstan, – 2018. – P. 188-189

88 Altybay A. On numerical simulations of the 1d wave equation with a distributional coefficient. comparison of the cases with neumann and dirichlet boundary conditions // XLII Международной научно-практической конференции на тему: «Инновационные технологии на транспорте: образование, наука, практика». – 2018. – Vol. 2. – P. 323-324

89 Altybay A. Numerical simulation of tsunami equation and GPU computing // International Scientific Conference of Students and Young Scientists «Farabi alemi», Almaty, Kazakhstan. – 2020. – P. 10-11.

APPENDIX A

Program code of Yaneko Method

```
void YanekoMethod_mpi(double *a, double *b, double *c, double *f, double *y, int n)
{
int size, rank, i1, i2;
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
int i, j, k, kk, m = n / size;
cout << " m: " << m << endl;
double  *ai = new double[n + 1], *bi = new double[n + 1], *ci = new double[n + 1], *fi = new double[n + 1];
double  *alfa = new double[n + 1], *betta_u = new double[n + 1], *betta_w = new double[n + 1], *betta_v = new double[n + 1];
double *u = new double[n + 1], *v = new double[n + 1], *w = new double[n + 1], *f1 = new double[n + 1], *f2 = new double[n + 1], *f3 = new double[n + 1];
double *a_u = new double[m], *a_v = new double[m], *a_w = new double[m], *y_rez = new double[m];
        double *alfa2 = new double[size], *betta2 = new double[size];
double *A = new double[size + 1], *B = new double[size + 1], *C = new double[size + 1], *D = new double[size + 1], *Z = new double[size + 1];
if (rank == 0)
{
for (i = 0;i < n;i++)
{
ai[i] = a[i];
ci[i] = c[i];
bi[i] = b[i];
fi[i] = f[i];
f3[i] = f[i];
}
for (i = 0;i <= n;i++)
{
                                        f1[i] = 0;
                                        f2[i] = 0;
}
                                    f3[n] = f3[n - 1];
                                    fi[n] = fi[n - 1];
                                        ci[0] = 0;
                                        ci[n] = 0;
                                        ai[0] = 0;
```

```
                    ai[n] = 0;
                    bi[0] = 1;
                    bi[n] = 1;
                } //end rank==0

i1 = (n*rank) / size;
i2 = (n*(rank + 1)) / size;
    MPI_Bcast(ai, n + 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(bi, n + 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(ci, n + 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(f1, n + 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(f2, n + 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(f3, n + 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(fi, n + 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
                    f1[i1] = 1;
                    f2[i2] = 1;
                    f3[i1] = 0;
                    f3[i2] = 0;
                    b[i1] = 1;
                    c[i1] = 0;
                alfa[i1] = -c[i1] / b[i1];
                betta_u[i1] = f1[i1] / b[i1];
                betta_v[i1] = f2[i1] / b[i1];
                betta_w[i1] = f3[i1] / b[i1];
for (i = i1 + 1;i <= i2;i++)
{

alfa[i] = -c[i] / (b[i] + a[i] * alfa[i - 1]);
betta_u[i] = (-a[i] * betta_u[i - 1] + f1[i]) / (b[i] + a[i] * alfa[i - 1]);
betta_v[i] = (-a[i] * betta_v[i - 1] + f2[i]) / (b[i] + a[i] * alfa[i - 1]);
betta_w[i] = (-a[i] * betta_w[i - 1] + f3[i]) / (b[i] + a[i] * alfa[i - 1]);
}

                u[i1] = 1; v[i1] = 0; w[i1] = 0;
                u[i2] = 0; v[i2] = 1; w[i2] = 0;
for (i = i2 - 1;i>i1;i--)
{
u[i] = (alfa[i] * u[i + 1] + betta_u[i]);
v[i] = (alfa[i] * v[i + 1] + betta_v[i]);
w[i] = (alfa[i] * w[i + 1] + betta_w[i]);
}
```

```
        kk = 0;
        for (i = i1;i <= i2;i++)
        {
        a_u[kk] = u[i];
        a_v[kk] = v[i];
        a_w[kk] = w[i];
        kk++;
        }
        MPI_Gather(a_u, m, MPI_DOUBLE, u, m, MPI_DOUBLE, 0,
MPI_COMM_WORLD);
        MPI_Gather(a_v, m, MPI_DOUBLE, v, m, MPI_DOUBLE, 0,
MPI_COMM_WORLD);
        MPI_Gather(a_w, m, MPI_DOUBLE, w, m, MPI_DOUBLE, 0,
MPI_COMM_WORLD);
        MPI_Bcast(u, n + 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
        MPI_Bcast(v, n + 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
        MPI_Bcast(w, n + 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
        if (rank == 0)
        {
        B[0] = b[0] + c[0] * u[1];
        C[0] = c[0] * v[1];
        D[0] = f[0] - c[0] * w[1];
        A[size] = a[size*m] * u[size*m - 1];
       B[size] = b[size*m] + a[size*m] * v[size*m - 1];
        D[size] = f[size*m] - a[size*m] * w[size*m - 1];
        for (j = 1;j<size;j++)
        {
        A[j] = a[j*m] * u[j*m - 1];
        B[j] = a[j*m] * v[j*m - 1] + b[j*m] + c[j*m] * u[j*m + 1];
        C[j] = c[j*m] * v[j*m + 1];
        D[j] = f[j*m] - a[j*m] * w[j*m - 1] - c[j*m] * w[j*m + 1];
        }
        alfa2[0] = -C[0] / B[0];
        betta2[0] = D[0] / B[0];
        for (i = 1;i<size;i++)
        {
        alfa2[i] = -C[i] / (B[i] + A[i] * alfa2[i - 1]);
        betta2[i] = (-A[i] * betta2[i - 1] + D[i]) / (B[i] + A[i] * alfa2[i - 1]);
        }
Z[size] = (D[size] - A[size] * betta2[size - 1]) / (B[size] + A[size] * alfa2[size - 1]);
        for (j = size - 1;j >= 0;j--)
```

```
        {
        Z[j] = (alfa2[j] * Z[j + 1] + betta2[j]);
        }
        }
          MPI_Bcast(Z, size + 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
        y[i1] = Z[rank];
        y[i2] = Z[rank + 1];
        for (i = i1 + 1;i<i2;i++)
        {
        y[i] = Z[rank] * u[i] + Z[rank + 1] * v[i] + w[i];
        }
        kk = 0;
        for (i = i1;i<i2;i++)
        {
        y_rez[kk] = y[i];
        kk++;
        }
        MPI_Gather(y_rez, m, MPI_DOUBLE, y, m, MPI_DOUBLE, 0,
MPI_COMM_WORLD);
        if (rank == 0)
        {
        cout << "\n Yanenko result  : \n";
        for (i=0;i<n;i++)
        cout<<y[i] << " ";
        }
        delete[] alfa, betta_u, betta_w, betta_v;
        delete[] u, v, w, f1, f2, f3;
        delete[] a_u, a_v, a_w, y_rez;
        delete[] alfa2, betta2;
        delete[] A, B, C, D, Z;
        }  //End ThomasMethod
```

Appendix B
Program code of software complex

```
class OOP():
    def __init__(self):
        self.win = tk.Tk()
        self.win.title("wave simulator")
        self.win.geometry('1024x600')
        self.createWidgets()
    def createWidgets(self):
        tabControl = ttk.Notebook(self.win)
        tab1 = ttk.Frame(tabControl)
        tab2 = ttk.Frame(tabControl)
        tabControl.add(tab1, text='1D case')
        tabControl.add(tab2, text='2D case')
        tabControl.pack(expand=1, fill="both")
        self.monty0 = ttk.LabelFrame(tab1, text='Choose singular coefficents')
        self.monty0.grid(row=0, column=0,  columnspan=6, padx=4,  pady=4)
        self.var1 = tk.IntVar()
        self.var2 = tk.IntVar()
        self.var3 = tk.IntVar()
        self.var4 = tk.IntVar()
        ttk.Checkbutton(self.monty0,
text="a(t)",variable=self.var1,width=10,offvalue=0).grid(row=0, column=0)
        ttk.Checkbutton(self.monty0,
text="b(t)",variable=self.var2,width=10).grid(row=0, column=1)
        ttk.Checkbutton(self.monty0,
text="c(t)",variable=self.var3,width=10).grid(row=0, column=2)
        ttk.Checkbutton(self.monty0,
text="d(t)",variable=self.var4,width=10).grid(row=0, column=3)
        self.monty1 = ttk.LabelFrame(tab1, text='Set domain size')
        self.monty1.grid(row=1,  rowspan=2,  column=0,   columnspan=6,  padx=4,
pady=4)
        ttk.Label(self.monty1,  text="Domain  legth,  Lx=").grid(row=1,  column=0,
pady=4)
        self.lx_f=ttk.Entry(self.monty1,width=11)
        self.lx_f.grid(row=1, column=1)
        self.lx_f.insert(10, 20.0)
        ttk.Label(self.monty1,  text="Time  legth,  Lt=").grid(row=1,  column=2,
pady=4)
        self.lt_f=ttk.Entry(self.monty1,width=11)
```

```
        self.lt_f.grid(row=1, column=3)
        self.lt_f.insert(10, 2.0)

        ttk.Label(self.monty1, text="Domain step, h=").grid(row=2, column=0,
pady=4)
        self.h_f=ttk.Entry(self.monty1,width=11)
        self.h_f.grid(row=2, column=1)
        self.h_f.insert(10, 0.05)
        ttk.Label(self.monty1, text="Time step, tau=").grid(row=2, column=2,
pady=4)
        self.t_f=ttk.Entry(self.monty1,width=11)
        self.t_f.grid(row=2, column=3)
        self.t_f.insert(10, 0.025)
        self.monty2 = ttk.LabelFrame(tab1, text='Set parametrs')
        self.monty2.grid( row=3, column=0, columnspan=6, padx=4,  pady=4)
        ttk.Label(self.monty2, text="Eps").grid(row=3, column=0,  pady=4)
        self.ep_f=ttk.Entry(self.monty2,width=10)
        self.ep_f.grid(row=3, column=1)
        self.ep_f.insert(10, 0.1)
        ttk.Label(self.monty2, text="Delta").grid(row=3, column=2,  pady=4)
        self.d_f=ttk.Entry(self.monty2,width=10)
        self.d_f.grid(row=3, column=3)
        self.d_f.insert(10, 1.0)
        ttk.Label(self.monty2,          text="Center").grid(row=3,          column=4,
pady=4,padx=4)
        self.c_f=ttk.Entry(self.monty2,width=10)
        self.c_f.grid(row=3, column=5)
        self.c_f.insert(10, 5)
        self.monty3 = ttk.LabelFrame(tab1, text='Set exponential parametrs')
        self.monty3.grid(row=4, column=0, columnspan=6, padx=4,  pady=4)
        ttk.Label(self.monty3, text="Sb").grid(row=4, column=0,  pady=4)
        self.Sb_f=ttk.Entry(self.monty3,width=11)
        self.Sb_f.grid(row=4, column=1,padx=8)
        self.Sb_f.insert(10, 1.5)
        ttk.Label(self.monty3, text="Sc").grid(row=4, column=2,  pady=4)
        self.Sc_f=ttk.Entry(self.monty3,width=11)
        self.Sc_f.grid(row=4, column=3,padx=8)
        self.Sc_f.insert(10, 1.5)
        ttk.Label(self.monty3, text="Sd").grid(row=4, column=4,  pady=4,padx=8)
        self.Sd_f=ttk.Entry(self.monty3,width=11)
        self.Sd_f.grid(row=4, column=5)
```

```
            self.Sd_f.insert(10, 1.5)
            self.monty4 = ttk.LabelFrame(tab1, text='Status screen')
            self.monty4.grid(row=5, column=0,  columnspan=6, padx=4,  pady=4)
            scrollb = ttk.Scrollbar(self.monty4)
            scrollb.grid(row=5, column=0, columnspan=4, pady=4)
            self.listbox = tk.Listbox(self.monty4,width=50)
            self.listbox.grid(row=5, column=0, columnspan=4, pady=2, padx=8)
            self.listbox.config(yscrollcommand=scrollb.set)
            scrollb.config(command=self.listbox.yview)
            self.monty5 = ttk.LabelFrame(tab1, text='Control bar')
            self.monty5.grid(row=6, column=0,  columnspan=6, padx=4,  pady=4)
            ttk.Button(self.monty5,text="Start",
width=25,command=self.esepte).grid(row=6, column=0, columnspan=3)
            ttk.Button(self.monty5,  text='Exit',  width=25,  command=quit).grid(row=6,
column=3,columnspan=3)
           self.monty6 = ttk.LabelFrame(tab1, text='Equation',width=512)
            self.monty6.grid(row=0,column=6, columnspan=3, pady=4)
            load = Image.open("wave1d.jpg")
            render = ImageTk.PhotoImage(load)
            img = ttk.Label(self.monty6, image=render)
            img.image = render
            img.grid(row=0, column=6,columnspan=3,rowspan=2)
            self.monty7 = ttk.LabelFrame(tab1, text='Result windows')
            self.monty7.grid(row=1,column=6,    columnspan=2,    padx=4,        pady=4,
rowspan=5)
            self.c = tk.Canvas(self.monty7, width = 600, height = 400, bg = 'white')
            self.c.grid(row=1, column=6, columnspan=2, rowspan=5)
            self.monty8 = ttk.LabelFrame(tab1, text='  ')
            self.monty8.grid(row=6,  column=6,  padx=4,  pady=4)
            ttk.Button(self.monty8,text="Save                                                graphic",
width=25,command=self.esepte).grid(row=6, column=6)
            ttk.Button(self.monty8,          text='Save          animation',          width=25,
command=quit).grid(row=6, column=7)
            #________2D_____________________________
            self.tab2_0 = ttk.LabelFrame(tab2, text='Choose singular coefficents')
            self.tab2_0.grid(row=0, column=0,  columnspan=6, padx=4,  pady=4)
            self.va1 = tk.IntVar()
            self.va2 = tk.IntVar()
            self.va3 = tk.IntVar()
            self.va4 = tk.IntVar()
```

```
            ttk.Checkbutton(self.tab2_0,
text="a(t)",variable=self.va1,width=10,offvalue=0).grid(row=0, column=0)
            ttk.Checkbutton(self.tab2_0,
text="b(t)",variable=self.va2,width=10).grid(row=0, column=1)
            ttk.Checkbutton(self.tab2_0,
text="c(t)",variable=self.va3,width=10).grid(row=0, column=2)
            ttk.Checkbutton(self.tab2_0,
text="d(t)",variable=self.va4,width=10).grid(row=0, column=3)
            self.tab2_1 = ttk.LabelFrame(tab2, text='Set domain size')
            self.tab2_1.grid(row=1, rowspan=2, column=0, columnspan=6, padx=4,
pady=4)
            ttk.Label(self.tab2_1, text="Domain legth, Lx=Ly=").grid(row=1, column=0,
pady=4)
            self.lx_f=ttk.Entry(self.tab2_1,width=10)
            self.lx_f.grid(row=1, column=1)
            self.lx_f.insert(10, 100.0)
            ttk.Label(self.tab2_1, text="Time legth, Lt=").grid(row=1, column=2,
pady=4)
            self.lt_f=ttk.Entry(self.tab2_1,width=10)
            self.lt_f.grid(row=1, column=3)
            self.lt_f.insert(10, 5.0)
            ttk.Label(self.tab2_1, text="Domain step, h=").grid(row=2, column=0,
pady=4)
            self.h_f=ttk.Entry(self.tab2_1,width=10)
            self.h_f.grid(row=2, column=1)
            self.h_f.insert(10, 0.5)
            ttk.Label(self.tab2_1, text="Time step, tau=").grid(row=2, column=2,
pady=4)
            self.t_f=ttk.Entry(self.tab2_1,width=10)
            self.t_f.grid(row=2, column=3)
            self.t_f.insert(10, 0.5)
            self.tab2_2 = ttk.LabelFrame(tab2, text='Set parametrs')
            self.tab2_2.grid( row=3, column=0, columnspan=6, padx=4,  pady=4)
            ttk.Label(self.tab2_2, text="Eps").grid(row=3, column=0,  pady=4)
            self.ep_f=ttk.Entry(self.tab2_2,width=10)
            self.ep_f.grid(row=3, column=1)
            self.ep_f.insert(10, 0.1)
            ttk.Label(self.tab2_2, text="Delta").grid(row=3, column=2,  pady=4)
            self.d_f=ttk.Entry(self.tab2_2,width=10)
            self.d_f.grid(row=3, column=3)
            self.d_f.insert(10, 1.0)
```

```
            ttk.Label(self.tab2_2,          text="Center").grid(row=3,          column=4,
pady=4,padx=4)
            self.c_f=ttk.Entry(self.tab2_2,width=10)
            self.c_f.grid(row=3, column=5)
            self.c_f.insert(10, 5)
            self.tab2_3 = ttk.LabelFrame(tab2, text='Set exponential parametrs')
            self.tab2_3.grid(row=4, column=0, columnspan=6, padx=4,  pady=4)
            ttk.Label(self.tab2_3, text="Sb").grid(row=4, column=0,  pady=4)
            self.Sb_f=ttk.Entry(self.tab2_3,width=11)
            self.Sb_f.grid(row=4, column=1,padx=8)
            self.Sb_f.insert(10, 1.5)
            ttk.Label(self.tab2_3, text="Sc").grid(row=4, column=2,  pady=4)
            self.Sc_f=ttk.Entry(self.tab2_3,width=11)
            self.Sc_f.grid(row=4, column=3,padx=8)
            self.Sc_f.insert(10, 1.5)
             ttk.Label(self.tab2_3, text="Sd").grid(row=4, column=4,  pady=4,padx=8)
            self.Sd_f=ttk.Entry(self.tab2_3,width=11)
            self.Sd_f.grid(row=4, column=5)
            self.Sd_f.insert(10, 1.5)
            self.tab2_4 = ttk.LabelFrame(tab2, text='Status screen')
            self.tab2_4.grid(row=5, column=0,  columnspan=6, padx=4,  pady=4)
            self.listbox2 = tk.Listbox(self.tab2_4,width=50)
            self.listbox2.grid(row=5, column=0, columnspan=4, pady=2, padx=8)
            self.listbox2.config(yscrollcommand=scrollb.set)
            self.tab2_5 = ttk.LabelFrame(tab2, text='Control bar')
            self.tab2_5.grid(row=6, column=0,  columnspan=6, padx=4,  pady=4)
            ttk.Button(self.tab2_5,text="Start",
width=25,command=self.esepte2).grid(row=6, column=0, columnspan=3)
            ttk.Button(self.tab2_5,  text='Exit',  width=25,  command=quit).grid(row=6,
column=3,columnspan=3)
            self.tab2_6 = ttk.LabelFrame(tab2, text='Equation',width=512)
            self.tab2_6.grid(row=0,column=6, columnspan=3, pady=4)
            load = Image.open("wave2d.jpg")
            render = ImageTk.PhotoImage(load)
            img = ttk.Label(self.tab2_6, image=render)
            img.image = render
            img.grid(row=0, column=6,columnspan=3,rowspan=2)
            self.tab2_7 = ttk.LabelFrame(tab2, text='Result windows')
            self.tab2_7.grid(row=1,column=6,    columnspan=2,    padx=4,        pady=4,
rowspan=5)
            self.c = tk.Canvas(self.tab2_7, width = 600, height = 400, bg = 'white')
```

```
        self.c.grid(row=1, column=6, columnspan=2, rowspan=5)
        self.tab2_8 = ttk.LabelFrame(tab2, text='  ')
        self.tab2_8.grid(row=6,  column=6,  padx=4,  pady=4)
        ttk.Button(self.tab2_8,text="Save                                    graphic",
width=25,command=self.esepte2).grid(row=6, column=6)
        ttk.Button(self.tab2_8,          text='Save          animation',          width=25,
command=quit).grid(row=6, column=7)
```