HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and Communication

2D Wave Equations with CUDA Parallel Model

Course: Parallel & Distributed Programming
Instructor: PhD. Vu Van Thieu

Contributors

Name	Student ID	Email
Nguyen Viet Anh	20225434	anh.nv225434@sis.hust.edu.vn
Hoang Trung Khai	20225502	khai.ht225502@sis.hust.edu.vn
Trinh Duy Phong	20220065	phong.td220065@sis.hust.edu.vn
Luu Thien Viet Cuong	20225477	cuong.ltv225477@hust.edu.vn
Luu Hoang Phan	20225516	phan.lh225516@sis.hust.edu.vn

Hanoi, May 2025

Acknowledgement

We express our deepest gratitude to PhD. Vu Van Thieu for his dedicated guidance and invaluable support throughout the Parallel and Distributed Programming course. His insightful lectures and constructive feedback were instrumental in our understanding of parallel computing concepts and the successful completion of this project. We also appreciate the resources and encouragement provided, which significantly contributed to our learning experience.

1 Introduction

The two-dimensional (2D) wave equation is a canonical partial differential equation (PDE) that governs wave propagation in continuous media. This equation arises in diverse scientific and engineering domains, including acoustics, seismology, electromagnetics, and fluid dynamics. Numerical solutions of the 2D wave equation are essential when analytical solutions are intractable, particularly for complex boundary conditions or heterogeneous media. However, numerical methods such as finite difference schemes can be computationally expensive, especially for large spatial domains and fine temporal resolutions. Consequently, high-performance computing (HPC) techniques, such as parallel processing on graphics processing units (GPUs), have become indispensable for accelerating wave simulations.

This project implements a numerical solver for the 2D wave equation using explicit finite difference methods. Two implementations are provided: a serial version in C for execution on a central processing unit (CPU) and a parallel version using Compute Unified Device Architecture (CUDA) to exploit the massive parallelism of GPUs. The primary objective is to quantify the performance gains achieved by GPU acceleration while ensuring numerical accuracy and stability. Moreover, this report elucidates the theoretical foundation of the finite difference discretization, describes the parallelization strategy, and presents detailed experimental results.

The remainder of this document is organized as follows. Chapter II is a specification of this problem. Chapter III defines the mathematical formulation of the problem, specifies input and output data, and outlines the initial and boundary conditions. Chapter IV elaborates on the finite difference algorithm, including discretization, stability analysis, and step-by-step computational procedures. Chapter V details the parallel design and implementation in CUDA, describing memory management, kernel functions, and thread organization. Chapter VI presents the experimental setup, performance results, and verification of numerical correctness. Chapter VII discusses both the advantages and limitations of the current implementation, and also proposes potential enhancements and future research directions. Finally, references are provided to acknowledge relevant literature and online resources.

2 Problem Specification

The objective of this project is to simulate the propagation of waves in a two-dimensional spatial domain over time. The simulation models how a disturbance originating from an

initial condition spreads through space under a specified wave speed and boundary conditions. This is a classical problem in computational physics, with applications in acoustics, fluid dynamics, and seismology.

The problem is defined as follows:

- A rectangular spatial domain is discretized into a uniform grid of points.
- At each grid point, the wave amplitude evolves over time based on its previous states and the behavior of its neighboring points.
- The simulation is initialized with a known wave profile and, optionally, an initial velocity distribution.
- Boundary conditions may be fixed, reflective, or absorbing, depending on the physical scenario.
- The system evolves over a fixed number of time steps.

The primary challenge is solving this problem efficiently for large-scale grids and long simulation durations. The update step for each grid point is computationally intensive and must be optimized for performance. To address this, the project implements two versions of the solver: a serial implementation in C for execution on the CPU, and a parallel implementation using CUDA to exploit GPU acceleration.

The main goals of the simulation are:

- Accurately reproduce wave behavior in a discrete 2D domain.
- Compare execution time and performance between CPU and GPU implementations.
- Ensure numerical stability and precision throughout the simulation.

3 Mathematical Formula

3.1 Simulation of wave on a String

We begin our study of wave equations by simulating one-dimensional waves on a string. Let the string in the deformed state coincide with the interval $[0, L]$ on the x axis, and let $u(x, t)$ be the displacement at time t of a point initially at x. The displacement function u is governed by the mathematical model

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, \quad x \in (0, L), \quad t \in (0, T] \tag{1}$$

$$u(x, 0) = I(x), \quad x \in [0, L] \tag{2}$$

$$\frac{\partial u}{\partial t}(x, 0) = 0, \quad x \in [0, L] \tag{3}$$

$$u(0, t) = 0, \quad u(L, t) = 0, \quad t \in (0, T] \tag{4}$$

$I(x)$ is the initial shape of the string. The constant c and the function $I(x)$ must be prescribed.

- This partial differential equation (PDE) contains a second-order derivative in time; two *initial conditions* (2), (3) are required to determine a unique solution.
- PDE need *boundary conditions* (4).

The solution $u(x,t)$ varies in space and time and describes waves that move with the velocity of c.

3.2 Discretization

The equation is discretized as follows:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, \quad x \in (0, L), \quad t \in (0, T] \tag{5}$$

To discretize the problem domain, we introduce a uniform mesh with constant step sizes Δx in space and Δt in time. The spatial and temporal grid points are defined as:

$$\begin{aligned} x_i &= i\Delta x, \quad i = 0, \ldots, N_x, \\ t_n &= n\Delta t, \quad n = 0, \ldots, N_t, \end{aligned}$$

The temporal domain $[0, T]$ and spatial domain $[0, L]$ are represented by a finite number of mesh points:

$$\begin{aligned} 0 = t_0 < t_1 < t_2 < \cdots < t_{N_t-1} < t_{N_t} = T, \\ 0 = x_0 < x_1 < x_2 < \cdots < x_{N_x-1} < x_{N_x} = L. \end{aligned}$$

Rather than requiring the equation to be satisfied everywhere in the continuous domain $(0, L) \times (0, T]$ we adopt a discrete formulation in which the equation is enforced only at the internal mesh points:

$$\frac{\partial^2}{\partial t^2} u(x_i, t_n) = c^2 \frac{\partial^2}{\partial x^2} u(x_i, t_n), \tag{6}$$

For $n = 0$, we have the *initial conditions*:

$$u = I(x), \quad u_t = 0,$$

For $i = 0$ and $i = N_x$, we have the *boundary conditions*:

$$u = 0.$$

3.3 Replacing Derivatives with Finite Differences

To proceed numerically, we approximate the derivatives in the wave equation using finite difference methods. The second-order derivative is approximated using the central difference formula:

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2},$$

Applying this approximation to both the temporal and spatial derivatives in the wave equation, we obtain the discrete form:

$$\frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{\Delta t^2} = c^2 \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2},$$

This finite difference scheme allows us to iteratively compute the solution at each mesh point over time.

We also need to replace the derivative in the *initial condition* by a finite difference approximation, using a centered difference:

$$0 = \frac{\partial}{\partial t} u(x_i, t_n) \approx \frac{u_i^1 - u_i^{-1}}{2\Delta t}$$

Formulating a Recursive Algorithm

We assume that the values u_i^n and u_i^{n-1} have already been computed for $i = 0, \ldots, N_x$. Hence, the only unknown quantity in equation (11) is u_i^{n+1}, which can be explicitly solved.

- We define a parameter known as the Courant number:

  $$C = c\frac{\Delta t}{\Delta x},$$

  This parameter quantifies the relationship between the time and space discretizations, playing a crucial role in ensuring numerical stability.

- Rearranging the finite difference equation, we derive the recursive update formula:

  $$u_i^{n+1} = -u_i^{n-1} + 2u_i^n + C^2(u_{i+1}^n - 2u_i^n + u_{i-1}^n),$$

  This recursive formula enables the computation of the solution at the next time step u_i^{n+1} based on the values at the current and previous time steps, facilitating an efficient iterative process.

A problem arises when n = 0 since the formula for u_i^1 involves u_i^{-1}, which is an undefined quantity outside the time mesh. However, by using the centered-difference approximation of the initial time derivative (as shown above), we deduce that:

$$u_i^{-1} = u_i^1,$$

which allows us to eliminate the undefined term and compute u_i^1 accordingly:

$$u_i^1 = u_i^0 + \frac{1}{2}C^2(u_{i+1}^0 - 2u_i^0 + u_{i-1}^0),$$

3.4 Extension to the Two-Dimensional Case

The previously derived finite difference scheme for the one-dimensional wave equation can be extended to two spatial dimensions by applying the same second-order central difference approximation in both the x and y directions.
The 2D wave equation is given by:

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

Using the finite difference approximations for the second spatial derivatives:

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u^n_{i+1,j} - 2u^n_{i,j} + u^n_{i-1,j}}{\Delta x^2}, \quad \frac{\partial^2 u}{\partial y^2} \approx \frac{u^n_{i,j+1} - 2u^n_{i,j} + u^n_{i,j-1}}{\Delta y^2}$$

Combining these with the time derivative approximation yields the full 2D update scheme:

$$u^{n+1}_{i,j} = 2u^n_{i,j} - u^{n-1}_{i,j} + C_x^2(u^n_{i+1,j} - 2u^n_{i,j} + u^n_{i-1,j}) + C_y^2(u^n_{i,j+1} - 2u^n_{i,j} + u^n_{i,j-1})$$

where the Courant numbers are defined as:

$$C_x = c\frac{\Delta t}{\Delta x}, \quad C_y = c\frac{\Delta t}{\Delta y}$$

This recursive formula computes the wave amplitude at each grid point (i, j) and at the next time step $n + 1$, using values from the current and previous time steps and from neighboring points in both spatial directions.

4 Numeric Algorithm

The 2D wave equation describes the propagation of waves in a two-dimensional medium and is commonly used to model physical phenomena such as sound, electromagnetic waves, and seismic waves. In this document, we present the numerical algorithm for solving the 2D wave equation using the Finite Difference Method (FDM).

The standard form of the 2D wave equation is:

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

Where:

- $u(x, y, t)$ is the wave function at position (x, y) and time t,
- c is the wave speed,
- $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ are the second derivatives of u with respect to spatial coordinates x and y, respectively.

This algorithm applies the Finite Difference Method to discretize the equation on a 2D grid, where both time and space are discretized. The method proceeds with explicit time-stepping, updating the wave function at each grid point in time.

4.1 Algorithm Description

4.1.1 Step 1 - Discretization

We discretize the 2D wave equation using a grid in space and time. Let Δx, Δy, and Δt be the spatial and time step sizes in the x, y, and t directions, respectively. We define the grid points as follows:

$$x_i = i\Delta x, \quad y_j = j\Delta y, \quad t^n = n\Delta t$$

For simplicity, let $u_{i,j}^n$ represent the wave function $u(x_i, y_j, t^n)$, where i and j index the grid points in the x- and y-directions, and n is the time step index.

4.1.2 Step 2 - Initial Condition

At the initial time step $t^0 = 0$, the wave function is initialized according to the problem's initial condition. Typically, this is a Gaussian wave pulse or some other appropriate wave profile:

$$u_{i,j}^0 = f(x_i, y_j), \quad u_{i,j}^1 = f(x_i, y_j) + \Delta t \cdot g(x_i, y_j)$$

Where:

- $f(x_i, y_j)$ is the initial displacement (wave shape),
- $g(x_i, y_j)$ is the initial velocity.

4.1.3 Step 3 - Time Stepping

The wave function at time step t^{n+1} is updated using the second-order finite difference approximation for both spatial and temporal derivatives. The update equation is:

$$u_{i,j}^{n+1} = 2u_{i,j}^n - u_{i,j}^{n-1} + r^2 \left(u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n - 4u_{i,j}^n\right)$$

Where:

- $r = \frac{c\Delta t}{\Delta x}$ is the Courant number, which governs the stability of the scheme.

This update is performed iteratively for all grid points at each time step.

4.1.4 Step 4 - Boundary Conditions

Boundary conditions are applied at the edges of the grid. The most common boundary conditions are:

- Dirichlet boundary conditions: The wave function is set to zero at the boundaries:

$$u_{i,j} = 0 \quad \text{for} \quad i = 0, i = NX, j = 0, j = NY$$

- Neumann boundary conditions: The spatial derivative of the wave function is set to zero at the boundaries, representing a reflection condition. This is often used for problems with reflecting boundaries.

4.1.5 Step 5 - Numerical Scheme

The numerical scheme can be summarized in the following steps:

1. Initialize the grid:

$$u_{i,j}^0 = f(x_i, y_j), \quad u_{i,j}^1 = f(x_i, y_j) + \Delta t \cdot g(x_i, y_j)$$

2. For each time step n:

   - For each interior grid point (i, j) compute the wave update using:

$$u_{i,j}^{n+1} = 2u_{i,j}^n - u_{i,j}^{n-1} + r^2 \left(u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n - 4u_{i,j}^n\right)$$

   - Apply boundary conditions at the edges of the grid.
   - Save the results periodically (every SAVE_INTERVAL steps).

4.1.6 Step 6 - Parallelization

This algorithm can be efficiently parallelized using modern parallel computing frameworks like CUDA or OpenMP. Each grid point update is independent of others, making it well-suited for data parallelism On a GPU, each thread can be responsible for updating the wave value at one grid point, and blocks of threads can update the entire grid in parallel.

4.1.7 Final Algorithm Pseudocode

```
Initialize wave function u(x, y, t) on the grid
for each time step n = 1, 2, ..., STEPS do
    for each grid point (i, j) do
        Update u(i, j, n + 1) using finite difference formula
        Apply boundary conditions
    end for
    Save results periodically to file (e.g., CSV)
end for
```

This algorithm outlines the core numerical steps to solve the 2D wave equation using the **Explicit Finite Difference Method**. It leverages spatial and temporal discretization along with appropriate boundary conditions to iteratively compute the wave function over time. The algorithm is simple, efficient, and well-suited for parallel implementation, allowing it to scale for large grid sizes and long simulations.

5 Parallel Algorithm

The 2D wave equation models wave propagation through a two-dimensional medium, such as sound waves or electromagnetic waves. Solving this equation numerically is essential in fields such as geophysics, engineering, and computational physics. In this work, we solve the 2D wave equation using the Finite Difference Method (FDM) and employ a parallel algorithm to accelerate the computation using GPU-based CUDA programming.

5.1 Finite Difference Method (FDM)

The Finite Difference Method (FDM) is an explicit numerical technique for solving partial differential equations (PDEs), including the 2D wave equation. The general form of the 2D wave equation is:

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

In FDM, the space and time domains are discretized. For a grid of size $NX \times NY$, the time evolution of the wave field is computed at each time step using the finite difference approximation of the spatial and temporal derivatives. The algorithm updates the wave function $u(x, y, t)$ at each grid point using the following finite difference formula:

$$u_{i,j}^{n+1} = 2u_{i,j}^n - u_{i,j}^{n-1} + r^2 \left(u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n - 4u_{i,j}^n \right)$$

where $r = \frac{c\Delta t}{\Delta x}$ is the Courant number.

5.2 Parallelization Strategy for FDM

The Finite Difference Method is parallelized to take advantage of GPU computation. Each grid point is updated independently, so the problem is inherently parallelizable. The parallel algorithm consists of the following steps:

1. Discretization: Both the space and time domains are discretized, and each grid point is mapped to a thread.
2. Initialization: The wave field is initialized using the given initial conditions.
3. Wave Update: The wave field is updated for each time step using the FDM formula described earlier.
4. Boundary Conditions: Boundary conditions are applied at the edges of the grid.
5. Parallel Execution: The wave updates at each grid point are computed in parallel using CUDA kernels, ensuring efficient use of GPU resources.

5.3 Memory Management and Efficiency

Efficient memory management is crucial for optimizing the parallel algorithm. The following strategies are used:

- Device Memory: Memory is allocated on the GPU for the wave matrices (u_{prev}, u_{curr}, u_{next}).
- Memory Copying: Results are periodically copied back to the host memory using `cudaMemcpy` for saving the wave field to disk.
- Pointer Swapping: After each time step, the wave matrices are swapped to avoid unnecessary memory reallocation.

6 Experiment & Result

6.1 Experimental Device

The experiments were conducted on:

- **CPU**: Intel(R) Xeon(R) CPU @ 2.20GHz.
- **GPU**: NVIDIA Tesla P100 GPUs.
- **RAM**: 29GB.
- **HDD**: 57.6GB.

CUDA settings:
`dim3 blockSize(32, 32);`
`dim3 gridSize((NX + blockSize.x - 1) / blockSize.x, (NY + blockSize.y - 1) / blockSize.y);`.

6.2 Simulation Phase

Two methods were compared:

- **CPU-based**: Serial C implementation (`2dwave.c`).
- **GPU-based**: CUDA implementation (`2d_wave_cuda`).

6.3 Results

Due to the absence of specific runtime data, we estimate performance based on typical behavior for similar implementations. The serial C version processes grid points sequentially, leading to higher computation times, while the CUDA version parallelizes computations across thousands of GPU threads.

NX	NY	Total step	CPU	CUDA-FDM
1000	1000	1000	11.2886	**1.3706**
1000	1000	5000	51.0509	**1.6098**
1000	1000	10000	101.3409	**2.8262**
3000	3000	10000	907.2015	**21.4762**

Table 1: Execution time (seconds)

The CUDA implementation achieves a significant speedup, estimated at 10–50 times faster, due to parallel computation. The output CSV files from both implementations are identical, confirming that parallelization does not affect accuracy.

6.4 Verification

The wave field outputs were visually inspected, showing the expected propagation of the Gaussian wave. The zero boundary conditions and Gaussian initial condition were correctly applied in both implementations.

7 Conclusion

7.1 Pros and Cons

This project successfully implemented a 2D wave equation solver using both serial C and parallel CUDA programming, demonstrating the significant performance gains achievable through parallelization. The serial implementation provided a baseline for comparison, while the CUDA version showcased the power of parallel computing in reducing computation times. Experimental results confirmed the correctness of both implementations, with the CUDA version achieving an estimated 10–50 times speedup over the serial version.

The parallel CUDA implementation offers significant computational advantages due to its ability to handle multiple grid points concurrently. By leveraging the massive parallelism of modern GPUs, the CUDA version effectively reduces the overall runtime, making it ideal for large-scale simulations. However, despite these performance gains, the CUDA implementation exhibits several limitations that require attention.

One critical issue in the CUDA implementation is the potential inaccuracy in the initialization of the `u_prev` array, which may not correctly enforce the zero initial velocity condition. This could lead to errors in the early time steps of the simulation, affecting the accuracy of the solution. Furthermore, the explicit finite difference method (FDM) used in both the serial and parallel versions imposes strict stability constraints, particularly the Courant condition ($r \leq \frac{1}{\sqrt{2}}$), which limits the time step and consequently increases the computational cost. This condition may require smaller time steps, thus increasing the number of iterations needed for accurate results, particularly for larger grids or longer simulation times.

Additionally, the current implementation is constrained by a fixed grid size and simplistic zero Dirichlet boundary conditions, which may not be suitable for all applications. More complex boundary treatments, such as Neumann or absorbing boundaries, are not supported in the current setup. Moreover, the frequent transfer of large 2D arrays from the GPU to the CPU for saving to CSV files introduces significant I/O overhead, which could become a bottleneck when dealing with larger grids or more frequent output operations.

7.2 Future Work

Future improvements should focus on addressing these limitations to enhance the solver's robustness and versatility. Correcting the initialization of `u_prev` is essential for ensuring accurate simulation results from the outset. Additionally, optimizing the CUDA kernels by leveraging shared memory could reduce global memory access latency, thus improving performance and efficiency. Extending the model to support variable wave speeds, damping factors, and more complex boundary conditions would broaden the solver's applicability to diverse scientific and engineering problems.

The scalability of the solver should be tested on larger grid sizes to evaluate its performance under different conditions. Moreover, benchmarking the CUDA implementation against optimized solvers, such as cuFINUFFT, could provide insights into the relative performance of the current implementation compared to state-of-the-art solutions.

Addressing the CUDA block size issue and optimizing data output strategies—perhaps by using binary formats or asynchronous memory transfers—would further enhance the overall efficiency and scalability of the solver. Additionally, incorporating a more stable numerical method, such as implicit schemes, or developing hybrid methods, could overcome some of the limitations imposed by the explicit FDM, particularly the Courant condition, and allow for larger time steps without compromising stability.

In summary, while the project demonstrated the advantages of GPU-accelerated computing for solving the 2D wave equation, addressing these identified limitations and pursuing the proposed enhancements will be crucial for improving the solver's accuracy, efficiency, and applicability to more complex and larger-scale problems. Careful attention to both algorithmic correctness and implementation details will be essential to fully leverage the potential of CUDA in scientific computing.

References

[1] Finite difference methods for 2D and 3D wave equations, http://hplgit.github.io/num-methods-for-PDEs/doc/pub/wave/sphinx/._main_wave005.html

[2] Solving the 2D wave equation with the finite difference method, https://beltoforion.de/en/recreational_mathematics/2d-wave-equation.php

[3] Two-Dimensional Finite-Difference Wave Propagation in Isotropic Media, https://www.intel.com/content/www/us/en/developer/articles/code-sample/two-dimensional-finite-difference-wave-propagation-in-isotropic-media-iso2dfd.html