HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

2D Wave Equations with CUDA Parallel Model

Course: Parallel & Distributed Programming

GROUP 15:

Lưu Thiện Việt Cường	20225477
Nguyễn Việt Anh	20225434
Hoàng Trung Khải	20225502
Lưu Hoàng Phan	20225516
Trịnh Duy Phong	20220065

ONE LOVE. ONE FUTURE.

Table of contents

1. Problem Specification
2. Mathematical Formula
3. Parallel Programming
4. Experiment
5. Conclusion

Problem Specification

1. Introduction - Problem

- **Problem:** Simulate real-time propagation of waves in a 2D environment by updating wave displacement values at each grid point over successive time steps. The output should support smooth visualization suitable for real-time animation.

- **Challenges:** High computational cost for large grids requires efficient parallelization strategies to achieve the necessary runtime performance.

Mathematical Formula

2. Mathematical Formula

1 - Dimension Equation

Let $u(x,t)$ be the displacement at time t of a pointinitially at x .
The displacement function u is governed by the mathematical model

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

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

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

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

$I(x)$ 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 velocity *c*.

2. Mathematical Formula

Discretization

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

We introduce the **uniformly distributed constant mesh points**, with spacings Δt and Δx. We have that:

$$x_i = i\Delta x,\ i = 0, \ldots, N_x, \quad t_n = n\Delta t,\ n = 0, \ldots, N_t\,.$$

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

$$0 = t_0 < t_1 < t_2 < \cdots < t_{N_t - 1} < t_{N_t} = T\,.$$

$$0 = x_0 < x_1 < x_2 < \cdots < x_{N_x - 1} < x_{N_x} = L\,.$$

2. Mathematical Formula

Discretization

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

We relax the strict condition that equation holds everywhere in the space - time domain $(0, L) \times (0, T]$ to the requirement that the PDE is fulfilled at the interior mesh points only:

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

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** $u = 0$

2. Mathematical Formula

Replacing derivatives by finite differences

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

We approximate the derivatives using finite differences.

Central differences formula: $f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$

Applying this approximation to both the time and space derivatives:

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

2. Mathematical Formula

Formulating a recursive algorithm

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

- We introduced parameter: $C = c\frac{\Delta t}{\Delta x}$

  which also known as the *Courant number*.

- By rearranging the terms, we obtain the recursive update formula:

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

2. Mathematical Formula

Formulating a recursive algorithm

$$\boxed{u_i^{n+1} = -u_i^{n-1} + 2u_i^n + C^2\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right)}$$

A problem with arises when n = 0 since the formula for u_i^1 involves u_i^{-1}

But we have $0 = \frac{\partial}{\partial t}u(x_i, t_n) \approx \frac{u_i^1 - u_i^{-1}}{2\Delta t}$ *(Initial condition)*

=> $u_i^{n-1} = u_i^{n+1}, \quad i = 0, \ldots, N_x, \; n = 0$

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

2. Mathematical Formula

Applying in 2-Dimension Equation

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

Turn into this recursive function for implementation:

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

Numeric Algorithms

3. Numeric Algorithms

Initialize Variables:

- Initialize the grid for the wave displacement and velocity: *u(x, y, t)* and *v(x, y, t)* where *u* is the displacement and *v* is the velocity

Initial Conditions:

- Set the initial wave field *u(x, y, t=0)* (often zero or a specific initial condition).
- Set the initial velocity field *v(x, y, t=0)* (often zero).

Update Rule (Finite Difference):

- Use the central difference scheme for the spatial and time derivatives to update the wave field:

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

3. Numeric Algorithms

Boundary Conditions:

- Apply fixed or periodic boundary conditions depending on the problem setup. Typically, at the edges of the grid, the displacement u is set to 0 or some predefined value.

Parallelization (CUDA):

- Use CUDA kernels to parallelize the spatial updates, ensuring the 2D grid is processed efficiently on the GPU.

Parallel Algorithms

4. Parallel Algorithms

We solve the 2D wave equation using the **Finite Difference Method (FDM)**

We employ a parallel algorithm to accelerate the computation using GPU-based CUDA programming

Finite Difference Method (FDM)

- Discretize space and time
- Computes the wave function *u(x,y,t)* at each time step

Formula

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

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

Each grid point can be updated independently → Ideal for parallelization

4. Parallel Algorithms

Parallelization Strategy

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 FDM, depending on the method selected

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 par allel using CUDA kernels, ensuring efficient use of GPU resources

Experiment & Result

4. Experiment

Device

- **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);
```

4. Experiment

Evaluation Method

Two methods were compared:

- **CPU-based:** Serial C implementation
- **GPU-based:** CUDA implementation

We evaluate the stimulation under different grid sizes and varying numbers of time steps.
When compared, **faster** simulations mean **better** performance

4. Experiment

Execution Time Comparison Result in seconds

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**

CONCLUSION

5. Conclusion

Advantage

- Achieved ~ 10 – 30x speedup over the CPU (serial C) version.
- Demonstrated strong performance gains via parallelism.
- Suitable for large-scale simulations.

Limitation

- Initialization bug in u_{prev} can affect early simulation steps.
- Courant condition imposes strict stability limits (smaller time steps).
- Simple boundary conditions reduce flexibility.
- I/O bottlenecks due to frequent large array transfers between GPU and CPU

5. Conclusion

Future Work

- Fix initialization of u_{prev}.
- Use shared memory for better performance.
- Extend to variable wave speed, damping, and complex boundaries.
- Optimize data output and memory transfer strategies.

HUST

THANK YOU !