NTNU

Norwegian University of
Science and Technology

Department of Computer Science

TDT4200 - Parallel Programming

Exercise 1A

1 Introduction

This exercise will prepare you for the baseline code used for the graded assignments this year. You will implement a 1D numerical solution to the Wave Equation using the Finite Differences Method (FDM) with forward differences (meaning you iterate forward in time).

The Wave Equation is given by

$$\frac{\partial^2 u}{\partial t^2} = c^2 \cdot \frac{\partial^2 u}{\partial x^2}, \tag{1}$$

where

- u is the wave's amplitude
- t is time
- x is space
- c is the wave's propagation rate (celerity)

Equation 1 can be visualised as movement on a string or a wire like a jumping rope pulled up and down. In the 2D case, you can think of it as waves moving through the ocean. Figure 1 shows the initial state when the code is successfully implemented.

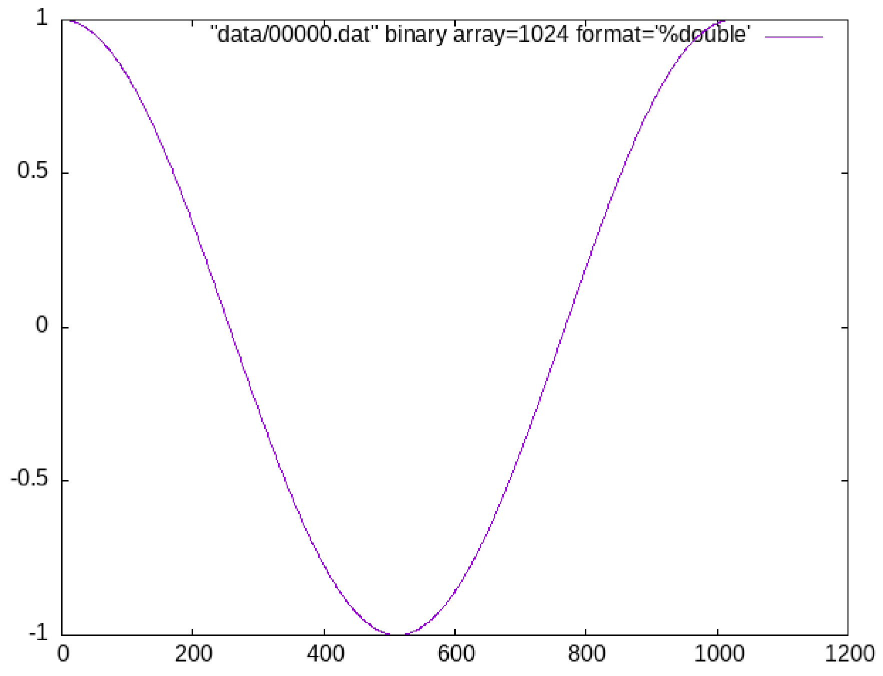

Figure 1: Image of starting condition to the system you will write in the programming part.

FDM is based on grinding the space and time domain with a finite set of points and approximating the values at the end of the solution by solving algebraic equations using nearby points. Figure 2 illustrates this where all the rows are in the same space at different times.

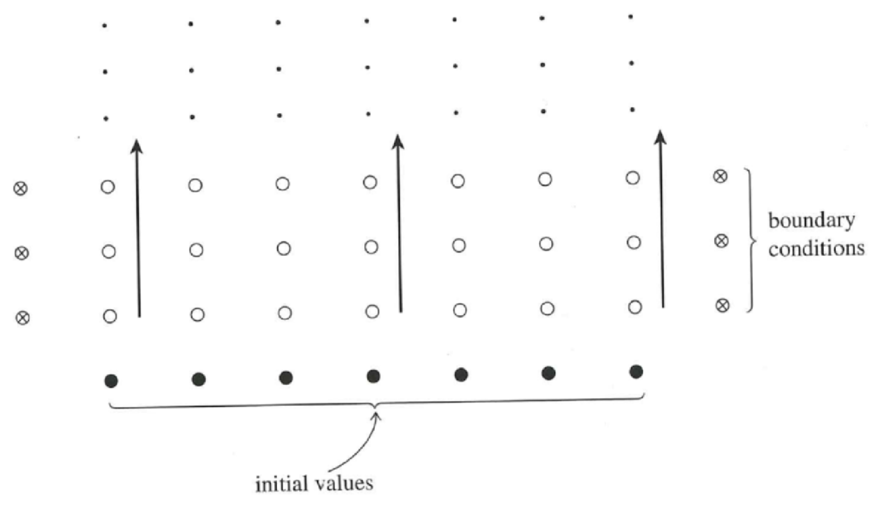

Figure 2: Finite Differences. The dots represent discrete points in space at different time steps. Remember, 1D is a line.

We will be using `central differences`. Let $\Delta x = h$, and substituting the central difference approximation

$$f''(x) \approx \frac{f(x-h) - 2 \cdot f(x) + f(x+h)}{h^2}$$

into Equation 1 for space yields

$$\frac{\partial^2 u}{\partial t^2} = c^2 \cdot \frac{u_{i-1}^t + u_{i+1}^t - 2u_i^t}{h^2}, \tag{2}$$

where u_a^b denotes $u(a \cdot \Delta x, b \cdot \Delta t) = u(x,t)$ for brevity.

Similarly, we substitute

$$f''(t) \approx \frac{f(t-\Delta t) + f(t+\Delta t) - 2 \cdot f(t)}{\Delta t^2}$$

into Equation 2 to obtain

$$\frac{u_i^{t-1} - 2 \cdot u_i^t + u_i^{t+1}}{\Delta t^2} = c^2 \cdot \frac{u_{i-1}^t + u_{i+1}^t - 2u_i^t}{h^2} \tag{3}$$

as the numerical equation.

2 Tasks

2.1 Programming

We use two time steps, the current and the previous time iteration, to approximate the next time iteration. All the tasks are marked and can be found in the file `wave_1d.c`. The attached `README.md` explains the `Makefile` and how to run the code. The code is buildable and runnable out-of-the-box, but it will only produce a single, empty `.dat` file in the `data` folder when run.

When the code is correctly implemented, the commands `make plot` and `make movie` will work as intended and create `.png` plots and a `.mp4` movie.

1. **Initialise the domain:**

   The first thing we need to do is to initialise the domain and make it ready for simulation. We apply

   $$u(x,0) = \cos(\pi x) \tag{4}$$

   as the initial condition of the system, where x is a point in space.

   The time step in the wave equation is usually derived from the Courant-Friedrichs-Lewy (CFL) condition to ensure stability in the numerical scheme. It is given by

   $$\Delta t \leq \frac{\Delta x}{c}. \tag{5}$$

   Implement the function `domain_initialize`. The function should:

   - Allocate memory for three sets of spatial domains: the previous, the current, and the next domain.
   - Apply the initial condition from Equation 4.
   - Set the time step, `dt`, based on Equation 5.

2. **Finalise the domain:**
   After allocating memory, we must also return it to the OS to prevent memory leaks.

   Implement the function `domain_finalize`. The function should free the memory allocated by `domain_initialize`.

3. **Move the buffer**

   In T1, you allocate memory for three sets of spatial domains in the buffer.

```
// Buffers for three time steps, indexed with two ghost points for the boundary.
real_t
    *buffers[3] = { NULL, NULL, NULL };
```

   Each time we do a time step, the oldest spatial domain is no longer needed, and the newly calculated spatial domain should be the current one. This is illustrated in Figure 3, where each square represents a space domain at a particular time step.

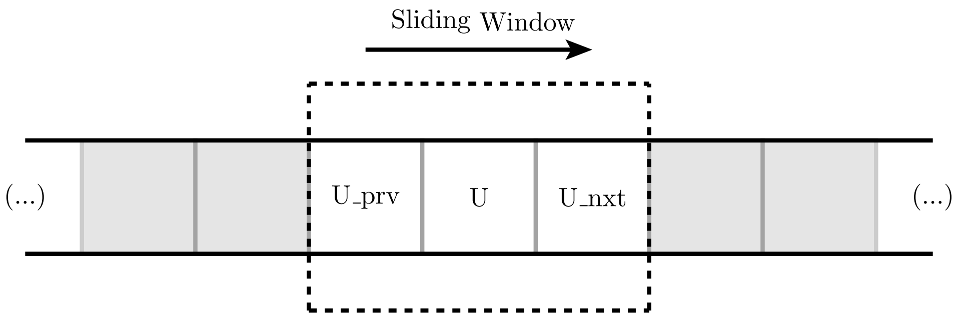

Figure 3: The window moves one time iteration to the right as a new state is calculated.

Define and implement a function that shifts the buffer domain window one step forward.

4. **Time Step**
   To integrate forward in time, we solve Equation 3 for u_i^{t+1} which yields

$$u_i^{t+1} = -u_i^{t-1} + 2u_i^t + \frac{\Delta t^2 c^2}{h^2} \cdot (u_{i-1}^t + u_{i+1}^t - 2u_i^t) \tag{6}$$

   for every point in space. Figure 4 illustrates the process.

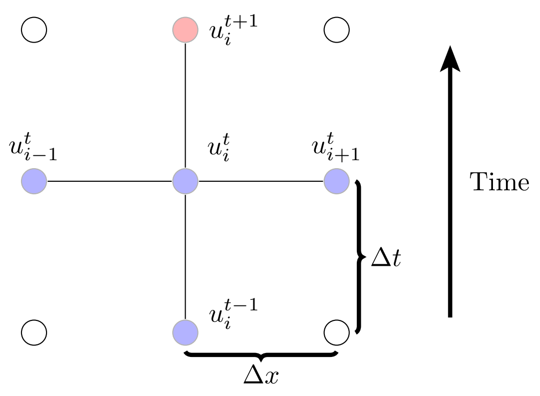

Figure 4: A visualisation of Equation 6. To calculate the new point (red), we use the already-known points (blue).

Define and implement a function that does one iteration of integration over the time domain (does one step forward in time). The function should calculate the next time iteration based on Equation 6 for every point in space.

5. **Boundary Conditions**
   We have added two ghost points to calculate the next point in space right next to the boundary. The boundary conditions are used when we are at the edge of our domain. In this solution, we use the Neumann (reflective) boundary. This means the ghost cell mirrors the point's value one step further into the space domain, as illustrated in Figure 5.

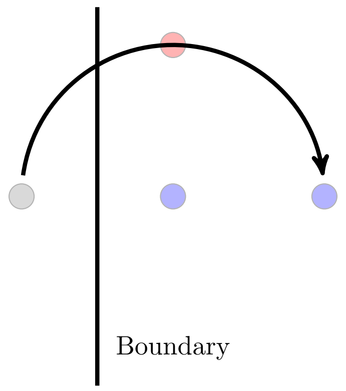

Figure 5: The ghost point outside the boundary (grey) copies the value of the cell one step inside the boundary as a reflection. This point is needed to calculate the next point in time (red)

Remember that there is a boundary on each side of the domain (left and right).

> Define and implement a function that sets the value of the ghost cell to mirror that of the point one further into the domain.

6. **Simulate**

   Now, it's time to combine the whole solution. In addition, we want to see how the wave progresses. Therefore, we want to save the state at the frequency given by the snapshot frequency.

   > Implement the function `simulate`. The function should for every time iteration:
   >
   > - Apply the boundary condition to the ghost cells (T5).
   > - Do one time step forward (T4).
   > - Store the domain in a `.dat` file if the time iteration matches that of the snapshot frequency, `snapshot_freq`, by using the `domain_save` function.

2.2 Theory questions

1. Explain the pros and cons of using macros like

```c
#define U_prv(i) buffers[0][(i)+1]
#define U(i)     buffers[1][(i)+1]
#define U_nxt(i) buffers[2][(i)+1]
```

   for simulations like the wave equation.

2. Mention at least one other boundary condition that could have been applied instead of the Neumann (reflective) boundary condition.
3. What happens if you don't allocate memory in T1?
4. What is the difference between

```c
float const *a;
```

   and

```c
float *const b;
```

   ?

3 Deliverables

Please deliver `.zip` file with the following contents containing your changes to the code:

- `wave_1d.c`
- `Makefile`
- A document with the answers to the theory questions.