

# Flood simulation

EduHPC'25 Peachy Assignment

## 1 Introduction

Students are provided with a sequential code that simulates the flow and accumulation of rainwater over a terrain. The program allows different scenarios to be selected through its input arguments. By default, it simulates the evolution of water on a  $30 \times 30$  km terrain.

The level of detail in each scenario can be adjusted by changing the number of cells into which the terrain is divided. The program arguments define the characteristics of either independent circular clouds or a cloud front, including their radius, direction, speed, and rainfall intensity. The simulation proceeds in one-minute steps. At each step, the clouds move and release a predetermined amount of water onto the cells below them. Each cell then transfers part of its water to neighbouring cells with lower water levels. More water is poured into cells with lower water levels than their neighbours. In this way, water flows to lower areas until the levels are equalised. Cells at the edge of the terrain also drain part of their water outside the simulation domain. This amount of water is removed from the simulation.

Figure 1 shows an example of the simulation in a valley scenario with a large dam.

The simulation continues either until a number of time steps, specified by a program argument, is completed, or until the amount of water transferred between cells falls below a given threshold, indicating that a sufficient equilibrium has been reached.

## 2 Sequential code details

### 2.1 Program arguments (mandatory)

These arguments define the basic simulation data, the terrain scenario, and a cloud front composed of multiple random clouds.

- `<rows>` Number of rows in the cell array.
- `<columns>` Number of columns in the cell array.

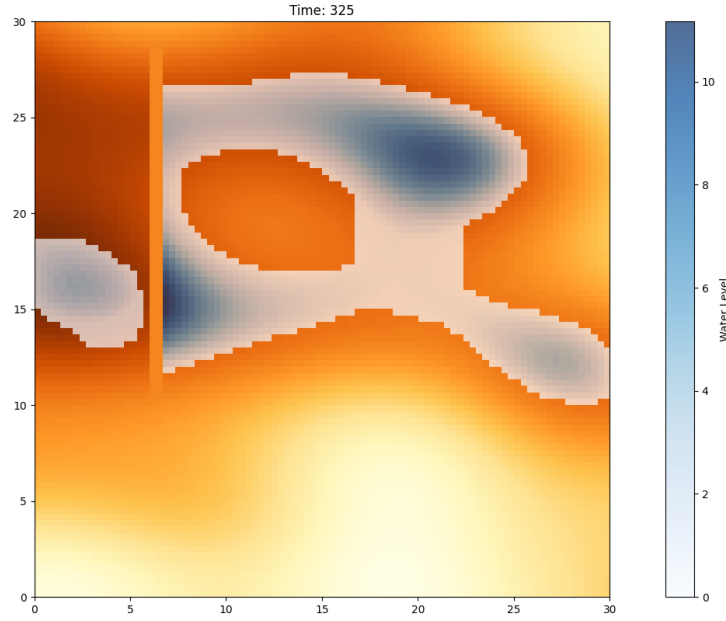


Figure 1: Example of the simulation in a valley scenario with a large dam.

- `<ground_scenario (M|V|D|d)>` The heights of the terrain cells are stored in an array. The program includes four predefined terrain scenarios, which are selected using a single-character code:
  - M: Mountain lakes.
  - V: Valley.
  - D: Valley with a dam at slightly higher elevations.
  - d: Valley with a dam at lower elevations.
- `<threshold>` The simulation stops when the highest amount of water discharged from one cell to another falls below this threshold.
- `<num_minutes>` The simulation stops after this number of minutes has been simulated.
- `<exaggeration_factor>` Multiplier for the rainfall discharge intensity. It allows for a less realistic but faster simulation. For example, a value of 60 means that in one minute, the amount of rainwater corresponding to one hour is discharged.
- `<front_distance>` Distance between the center of the simulation domain and the center of the cloud front.
- `<front_width>` Width of the cloud front.

- `<front_depth>` Depth of the cloud front. These two parameters (width and depth) define the dimensions of the rectangular region within which random clouds are generated. This region is rotated and translated based on the other parameters in order to correctly position the front, so that its direction leads the clouds into the domain.
- `<front_direction (degrees)>` Direction of the cloud front in degrees.
- `<num_random_clouds>` Number of clouds generated in the front.
- `<cloud_max_radius (km)>` Maximum radius of the clouds. For each cloud, a radius is generated randomly between this value and its half.
- `<cloud_max_intensity (mm/h)>` Maximum rainfall intensity. For each cloud, an intensity is generated randomly between this value and its half. Rainfall intensity is considered:
  - **Normal:** up to 15 mm/h,
  - **Heavy:** 15–30 mm/h,
  - **Very Heavy:** 30–60 mm/h,
  - **Torrential:** above 60 mm/h.
 Torrential rainfall exceeding 120–140 mm/h is rare but realistic.
- `<cloud_max_speed (km/h)>` Maximum speed. Each cloud is assigned a speed randomly between this value and its half.
- `<cloud_max_angle_aperture (degrees)>` Maximum aperture angle. Each cloud’s movement direction is randomly selected within the range defined by the front direction plus or minus half of this angle.
- `<clouds_rnd_seed>` Random seed used to reproduce the cloud generation in the front.

## 2.2 Optional program arguments

These arguments allow the user to specify an arbitrary number of individual clouds, in addition to those generated by the front. Each cloud is defined by a group of six parameters:

- `<cloud_start_x (km)>` Initial x-coordinate of the cloud centre.
- `<cloud_start_y (km)>` Initial y-coordinate of the cloud centre.
- `<cloud_radius (km)>` Radius of the cloud.
- `<cloud_intensity (mm/h)>` Rainfall intensity.

- `<cloud_speed (km/h)>` Cloud speed.
- `<cloud_angle (degrees)>` Direction of cloud movement, in degrees.

These arguments enable the creation of custom scenarios with different spatial and temporal dynamics, as well as varying computational demands in the different phases of the simulation.

### 3 Program output

At the end of the simulation, the program displays the execution time of the computation phase, excluding the initialisation time, along with several output values that help verify the correctness of the results. These include the total number of iterations; the iteration at which the highest amount of water was transferred between two cells; the maximum amount of water transferred in a single step; the highest water level reached in any cell; the total amount of rainwater discharged by the clouds; the total amount of water remaining on the ground at the end of the simulation; and the amount of water lost through the boundaries of the terrain.

### 4 Debug mode

If the program is compiled with the `-DDEBUG` flag (for example, using `make debug`), it activates sections of the code that print to the standard output the arguments read, the terrain height matrix, the list of generated clouds with their corresponding data, and, at each simulation step, the changes in the terrain water level matrix. This mode can be useful for detecting possible errors or for gaining a better understanding of the program's behaviour with very small test cases.

If the program is compiled with both `-DDEBUG` and `-DANIMATION` (e.g., using `make animation`), additional parts of the code are activated that write to the standard output the data and matrices required to generate an animation of the simulation. To do so, redirect the program output to a file and run the provided Python script (`animation.py`) with that file as input. The script first displays the terrain; after a key is pressed, it proceeds to generate the simulation. The resulting animation is saved as an MP4 file.

These are the arguments used in Figure 1 (which corresponds to a frame from the generated video):

```
./flood_seq 90 90 D 0.0000001 350 60 30 40 5 240 11 4 120 80
            45 667245 25 18 2 5 50 260
```

Students are expected to generate their own examples with significant computational loads in order to carry out performance and timing measure-

ments. It is recommended to verify that the program still produces correct results after each modification.

## **5 Part of the program to parallelise and optimise**

The source code explicitly indicates the sections, functions, and data structures that students are required to parallelise and optimise. These are the only parts of the code where modifications are allowed.

Code located outside the marked region—such as sections responsible for initialisation, performance measurement, result output, etc. must not be modified under any circumstances.

Within the modifiable region, students are allowed to introduce changes, perform optimisations, and modify or create new data structures, provided that the core logic and overall behaviour of the algorithm remain unchanged.

## **6 Target**

The objective is to parallelise and optimise the code without modifying the underlying algorithm or altering the simulation results. It is essential to identify which parts of the code can be parallelised directly, and which require modifications to resolve issues such as race conditions or data dependencies.

## **7 Compilation**

To measure execution times accurately, the program should be compiled using the compiler's highest optimisation level (e.g., `gcc -O3`). The focus is on implementing changes that do not interfere with, and ideally support, the compiler's own optimisation mechanisms.