

Blackout simulation

EduHPC'26 Blackout 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 30×30 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.

2 Introduction

Students are provided with a sequential code that simulates the operation of an electrical power grid composed of generation stations distributed over a two-dimensional mesh. The program allows different scenarios to be selected through its input arguments.

Each cell of the grid represents a generation station with a given production capacity. At every simulation step, each station must satisfy an electrical demand that follows a simplified daily consumption cycle. In addition, random demand spikes may occur at individual stations, increasing their load.

If the demand assigned to a station exceeds its production capacity, the station is disconnected for a fixed recovery period to protect the equipment. The unsatisfied demand is then redistributed among neighbouring stations that remain active. This redistribution may overload additional stations, producing cascading failures that can propagate through the network.

The simulation advances in discrete time steps. During each step, the demand at every station is updated, overloads are detected, disconnected stations are managed, and any resulting demand redistribution is performed. Stations whose recovery period has expired are automatically reconnected to the grid.

Figure 1 shows an example of the simulation for the single-city scenario. The colour scale represents the power produced by each station, while disconnected stations are highlighted in black.

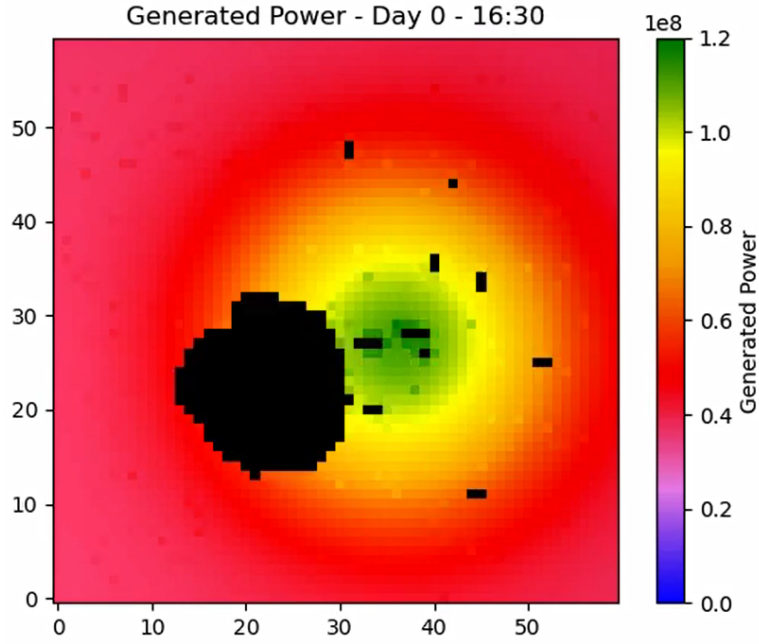


Figure 1: Example of the blackout simulation.

The simulation continues until the number of simulated minutes specified by the user has elapsed. Depending on the demand evolution and the occurrence of demand spikes, the network may remain stable or experience cascading failures that lead to widespread power outages.

3 Sequential code details

3.1 Program arguments

These arguments define the size of the electrical grid, the simulation duration, the distribution of generation capacity, the characteristics of the demand spikes, and the random seed used to reproduce the simulation.

- `<rows>` Number of rows in the grid.

- `<columns>` Number of columns in the grid.
- `<scenario (C|R)>` Distribution of the installed generation capacity across the grid:
 - C: Single-city scenario, with high generation capacity concentrated near the centre of the grid and decreasing towards the borders.
 - R: Multi-city scenario, with several independent regions of high generation capacity.
- `<simulation_start_minute>` Minute of the day at which the simulation starts.
- `<num_minutes>` Total number of simulated minutes.
- `<simulation_step>` Number of simulated minutes advanced in each iteration.
- `<k_cooldown>` Number of minutes that a station remains disconnected after an overload before it can reconnect.
- `<spike_center_frac>` Fraction of the day at which demand spikes are most likely to occur.
- `<spike_time_width_frac>` Width of the time window around the spike centre where demand spikes are more likely.
- `<spike_size_mean>` Mean relative size of demand spikes.
- `<spike_size_sigma>` Standard deviation of the relative spike size.
- `<spike_ratio>` Probability that a station generates a demand spike during a simulation step.
- `<rng_seed>` Random seed used to reproduce exactly the same simulation.

4 Program output

At the end of the simulation, the program displays the execution time of the computation phase, excluding the initialisation time, together with several output values that can be used to verify the correctness of the implementation. These include the simulation minute at which the highest station demand was observed; the value of that maximum demand; the simulation minute at which the largest unsatisfied demand occurred due to station disconnections; the value of that maximum unsatisfied demand; the total power

supplied by all stations throughout the simulation; the total unsatisfied demand accumulated during the simulation; and the accumulated standard deviation of the station power output histogram over all simulation steps.

5 Debug mode

If the program is compiled with the `-DDEBUG` flag (for example, using `make debug`), additional diagnostic information is printed to the standard output. This includes the generated station capacities and other intermediate data that can help verify the correctness of the simulation when using small test cases.

If the program is compiled with the `-DANIMATION` flag (e.g., using `make animation`), the program generates an output file containing the data required to animate the simulation. This file stores the initial station capacities together with the power produced and the operational status of every station at each simulation step. The provided Python script (`animation.py`) reads this file and generates an MP4 animation showing the evolution of the electrical grid during the simulation.

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

```
./blackout_seq 60 60 C 0 1440 1 5 0.77 0.05 0.1 0.3 0.1 6928461
```

Students are expected to generate their own examples with significant computational loads in order to carry out performance and timing measurements. It is recommended to verify that the program still produces correct results after each modification.

6 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 program where modifications are allowed.

Code located outside the marked region—including the sections responsible for argument processing, memory allocation, initialisation, performance measurement, and result output—must not be modified under any circumstances.

Within the modifiable region, students may introduce changes, perform optimisations, and modify or create new data structures, provided that the overall behaviour and output of the simulation remain unchanged.

7 Target

The objective of this assignment is to parallelise and optimise the provided sequential implementation without modifying the underlying simulation model or altering the numerical results. Students are expected to identify the computationally intensive parts of the program, determine which sections can be parallelised directly, and resolve any race conditions or data dependencies that arise during the parallelisation process.

8 Compilation

To obtain meaningful performance measurements, the program should be compiled using the compiler's highest optimisation level (e.g., `gcc -O3`). Students should implement optimisations that do not interfere with, and ideally complement, the compiler's own optimisation mechanisms.