COE 892 - Distributed Cloud Computing

Winter 2025

Dr. Khalid A. Hafeez

Lab 1 - Concurrency vs. Parallelism

Submission Date: February 2, 2025

Eric Muzzo

501019745

Introduction

This lab uses python to process the data of several land mine detector robots. The threading module in python is used to compare the execution time of 10 robots against the sequential execution time of the same programs. The program reads a map.txt file which contains both empty cells and mines that the robot must traverse through. Part 1 of this lab runs each robot through the map by executing a series of commands provided by the robot API. If a robot encounters a mine and does not execute the dig command, it is destroyed and execution halts. The second part of the lab involves digging and disarming mines by attempting to find a valid pin which, when concatenated with the serial number of the mine and hashed using the sha256 hash function, begins with 6 leading zeros by brute force.

Program Design & Overview

The implementation of the lab separates each object in the lab into their own classes. In my models.py file, I defined a class for the map, a map cell, and a rover. The use of each of these classes is detailed below.

Cell & Map Data Structures

To avoid complex calculations at rover runtime when determining the next cell to traverse to, I decided to represent my map as a fully linked data structure of Cell class instances. Each cell has attributes for its coordinates, value ("EMPTY" or "MINE"), serial number (applicable in part 2), and references to the left, right, up and down cells.
The Map class stores a 2D list of Cell objects. This class has methods for processing the map.txt and mines.txt files into my defined data structures. This involves performing the linking of each cell to its neighbours before considering the rovers.

Rover Data Structure

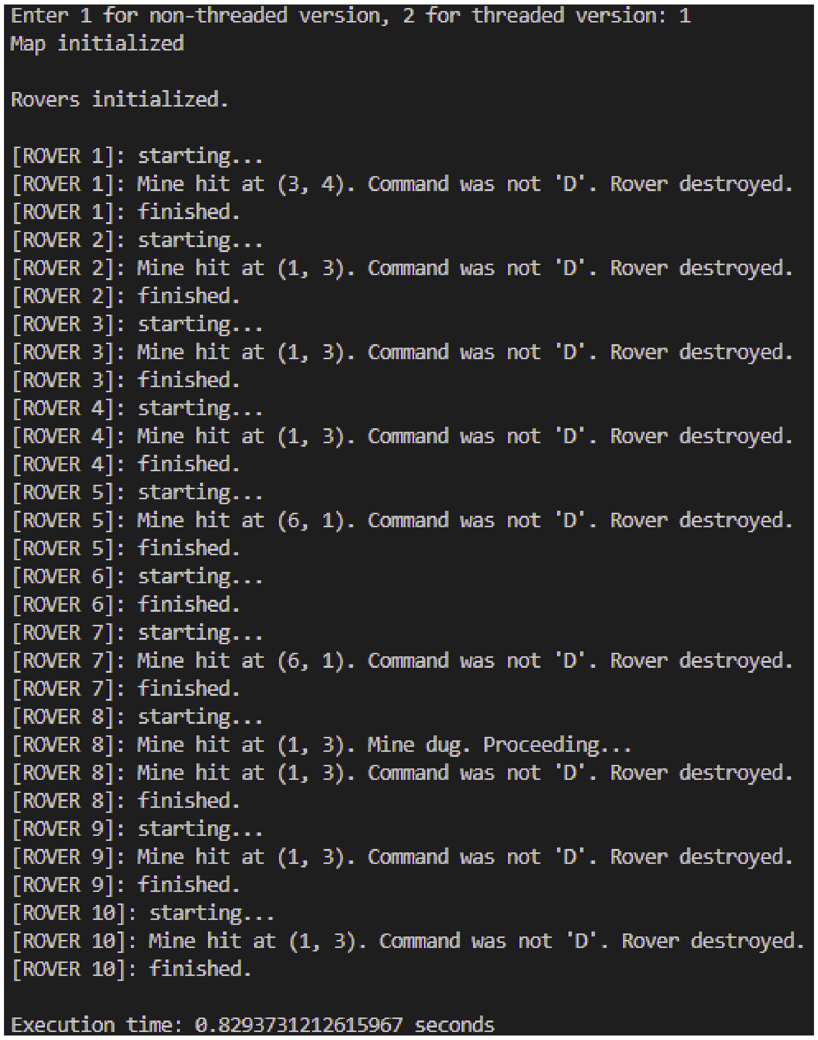
The Rover class represents an instance of a rover and stores attributes like id, a list of commands, its position (Cell reference) and orientation. I built class methods for running the rover through its commands (*run()*), executing a command (*move()*), and mining a mine (*mine()*).

Execution

In both parts of the lab, rather than duplicating the programs for sequential vs parallel execution, I just included a command line option to select which method of execution to run. The results are shown for each part of the lab below. I created an arbitrary 12 x 12 map with zeros representing empty cells and ones representing mines (See appendix for the map used in this demonstration). For part 2 of the lab where mines have serial numbers, when preprocessing the files I simply iterate through the mine serial numbers for each mine parsed in the map.txt file and assign it that serial number. This is of course stored in the Cell object.

Part 1

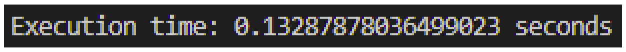
First we will execute a sequential approach. The execution time is shown below.



```
Enter 1 for non-threaded version, 2 for threaded version: 1
Map initialized

Rovers initialized.

[ROVER 1]: starting...
[ROVER 1]: Mine hit at (3, 4). Command was not 'D'. Rover destroyed.
[ROVER 1]: finished.
[ROVER 2]: starting...
[ROVER 2]: Mine hit at (1, 3). Command was not 'D'. Rover destroyed.
[ROVER 2]: finished.
[ROVER 3]: starting...
[ROVER 3]: Mine hit at (1, 3). Command was not 'D'. Rover destroyed.
[ROVER 3]: finished.
[ROVER 4]: starting...
[ROVER 4]: Mine hit at (1, 3). Command was not 'D'. Rover destroyed.
[ROVER 4]: finished.
[ROVER 5]: starting...
[ROVER 5]: Mine hit at (6, 1). Command was not 'D'. Rover destroyed.
[ROVER 5]: finished.
[ROVER 6]: starting...
[ROVER 6]: finished.
[ROVER 7]: starting...
[ROVER 7]: Mine hit at (6, 1). Command was not 'D'. Rover destroyed.
[ROVER 7]: finished.
[ROVER 8]: starting...
[ROVER 8]: Mine hit at (1, 3). Mine dug. Proceeding...
[ROVER 8]: Mine hit at (1, 3). Command was not 'D'. Rover destroyed.
[ROVER 8]: finished.
[ROVER 9]: starting...
[ROVER 9]: Mine hit at (1, 3). Command was not 'D'. Rover destroyed.
[ROVER 9]: finished.
[ROVER 10]: starting...
[ROVER 10]: Mine hit at (1, 3). Command was not 'D'. Rover destroyed.
[ROVER 10]: finished.

Execution time: 0.8293731212615967 seconds
```

Figure 1: Part 1 Sequential Execution Time

We then use option 2 to run the same program in parallel. The terminal output is not shown due to multiple threads battling for standard output, which makes the output quite messy.

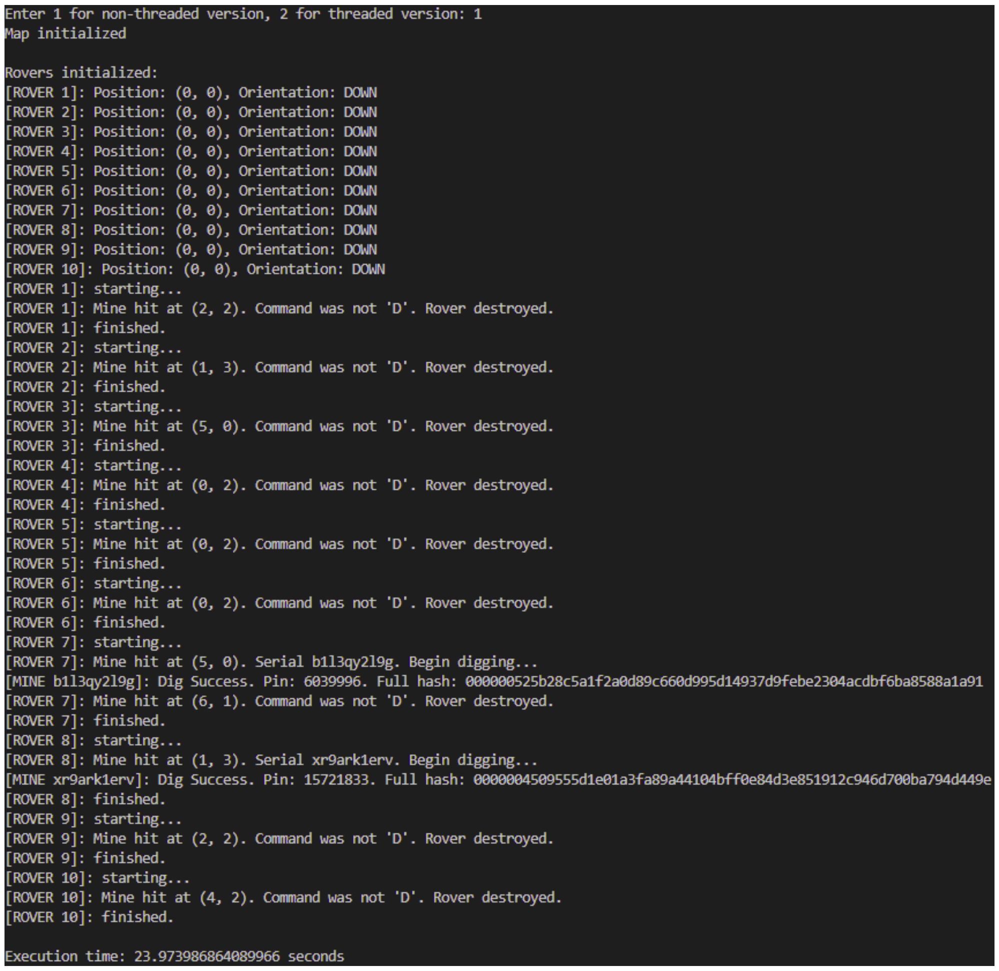


```
Execution time: 0.13287878036499023 seconds
```

Figure 2: Part 1 Parallel Execution Time

Part 2

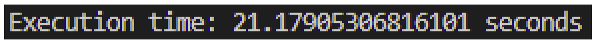
Again, we start with a sequential execution. In this demonstration, only rovers 7 and 8 hit mines and execute the dig command. The full output is shown below.



```
Enter 1 for non-threaded version, 2 for threaded version: 1
Map initialized

Rovers initialized:
[ROVER 1]: Position: (0, 0), Orientation: DOWN
[ROVER 2]: Position: (0, 0), Orientation: DOWN
[ROVER 3]: Position: (0, 0), Orientation: DOWN
[ROVER 4]: Position: (0, 0), Orientation: DOWN
[ROVER 5]: Position: (0, 0), Orientation: DOWN
[ROVER 6]: Position: (0, 0), Orientation: DOWN
[ROVER 7]: Position: (0, 0), Orientation: DOWN
[ROVER 8]: Position: (0, 0), Orientation: DOWN
[ROVER 9]: Position: (0, 0), Orientation: DOWN
[ROVER 10]: Position: (0, 0), Orientation: DOWN
[ROVER 1]: starting...
[ROVER 1]: Mine hit at (2, 2). Command was not 'D'. Rover destroyed.
[ROVER 1]: finished.
[ROVER 2]: starting...
[ROVER 2]: Mine hit at (1, 3). Command was not 'D'. Rover destroyed.
[ROVER 2]: finished.
[ROVER 3]: starting...
[ROVER 3]: Mine hit at (5, 0). Command was not 'D'. Rover destroyed.
[ROVER 3]: finished.
[ROVER 4]: starting...
[ROVER 4]: Mine hit at (0, 2). Command was not 'D'. Rover destroyed.
[ROVER 4]: finished.
[ROVER 5]: starting...
[ROVER 5]: Mine hit at (0, 2). Command was not 'D'. Rover destroyed.
[ROVER 5]: finished.
[ROVER 6]: starting...
[ROVER 6]: Mine hit at (0, 2). Command was not 'D'. Rover destroyed.
[ROVER 6]: finished.
[ROVER 7]: starting...
[ROVER 7]: Mine hit at (5, 0). Serial b1l3qy2l9g. Begin digging...
[MINE b1l3qy2l9g]: Dig Success. Pin: 6039996. Full hash: 000000525b28c5a1f2a0d89c660d995d14937d9febe2304acdbf6ba8588a1a91
[ROVER 7]: Mine hit at (6, 1). Command was not 'D'. Rover destroyed.
[ROVER 7]: finished.
[ROVER 8]: starting...
[ROVER 8]: Mine hit at (1, 3). Serial xr9ark1erv. Begin digging...
[MINE xr9ark1erv]: Dig Success. Pin: 15721833. Full hash: 0000004509555d1e01a3fa89a44104bff0e84d3e851912c946d700ba794d449e
[ROVER 8]: finished.
[ROVER 9]: starting...
[ROVER 9]: Mine hit at (2, 2). Command was not 'D'. Rover destroyed.
[ROVER 9]: finished.
[ROVER 10]: starting...
[ROVER 10]: Mine hit at (4, 2). Command was not 'D'. Rover destroyed.
[ROVER 10]: finished.

Execution time: 23.973986864089966 seconds
```

Figure 3: Part 2 Sequential Execution Time

Running the same program again with option 2 for parallel execution, we get:



```
Execution time: 21.17905306816101 seconds
```

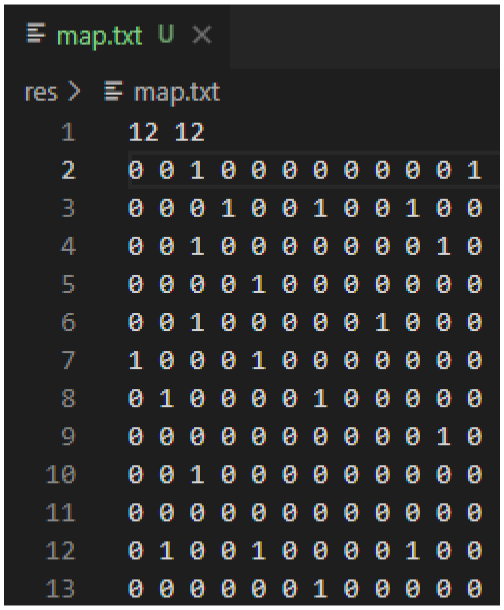
Figure 4: Part 2 Parallel Execution Time

Conclusion

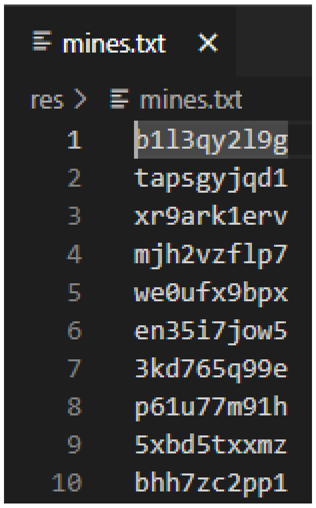
Part 1 demonstrates a clear improvement in execution time when run in parallel compared to a sequential execution time. This is likely due to the fetching of rover data from the API which takes the most amount of time. When we fetch these in parallel, each rover can begin executing while the rover initialization is awaiting an HTTP response for the next rover. Although the speedup factor seen in part 2 was not as dramatic, this would likely be more evident if more rovers hit mines and executed the dig command. To conclude, making use of threads in data processing reduces the idle time of CPU resources to complete a larger task in a smaller amount of time.

Appendix

map.txt & mines.txt



```
12 12
0 0 1 0 0 0 0 0 0 0 0 1
0 0 0 1 0 0 1 0 0 1 0 0
0 0 1 0 0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 1 0 0 0
1 0 0 0 1 0 0 0 0 0 0 0
0 1 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0
0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0 0 0 0 0
```



```
b1l3qy2l9g
tapsgyjqd1
xr9ark1erv
mjh2vzflp7
we0ufx9bpx
en35i7jow5
3kd765q99e
p61u77m91h
5xbd5txxmz
bhh7zc2pp1
```