CMPE 300 - Analysis of Algorithms
Project 2 Report

Mehmet Ali Özdemir
2021400000

December 23, 2024

Contents

1 Introduction

Provide a brief overview of the project.

This is a simulation of a battlefield consisting of units of 4 factions. The battlefield is partitioned in a checkered fashion and each part is operated by a worker thread. Workers exchange boundary data with their neighbors to maintain consistent information on the battlefield.

2 Design Decisions and Assumptions

Describe the major design decisions you made while implementing the project. Include any assumptions that were necessary to simplify or clarify the implementation.

Questions to Address:

- How did you structure the grid and the units?
- What partitioning strategy did you use, and why?
- How did you handle unit movements, attacks, and healing?

Firstly, I adopted the Object-Oriented Programming approach for factions. There are a Unit superclass and four faction classes. This approach makes the code more organized and maintainable. Each unit is an object of one of these four classes. The grid is a 2D array consisting either of a point string or a unit object.

I decided to use checkered partitioning because at the beginning I thought that striped partitioning might be easier to implement, so I challenged myself by selecting checkered partitioning. However, since implementing communication between threads in checkered partitioning struggled me too much, I think implementing communication between threads for striped partitioning would not be much easier than checkered partitioning.

The handling of unit movements was not easy. Firstly, moving an air unit in the same partitions is straightforward, but when it tries to move to sub-grid of another worker, it needs the sub-grid information of its neighbors. Implementing attacks was easier than implementing movements, but this may be because of implementing communication between threads in the movement phase and using the same logic. The implementation of healing phase was easy since it does not require sending or receiving information to or from the neighbors. It just checks whether the unit is attacked or skipped in that round.

3 Implementation Details

Detail the steps taken to implement the project, the challenges faced, and how they were overcome.

Firstly, I implemented faction classes. The Unit class is superclass of four factions and includes common properties and functions of each faction class. The faction classes contain faction-specific properties and functions. Then, I read the header data from the file. After that, MPI is initialized and grid is partitioned in a checkered fashion. Then, each sub-grid is sent to the corresponding workers. After that, the first wave starts, then the manager reads unit data of the first wave from the file and sends them to the corresponding workers. The workers receives these data and creates units in their sub-grids according to the position data that are sent to them. Then, the first round starts.

Firstly, each worker finds its neighbors and puts their sub-grids into a dictionary. Synchronizing means that every worker sends its sub-grid to their neighbors. Synchronizing is made in the code many times to make the data consistent, especially between the phases. Then, the movements for air units are computed and put into an array. Also, the workers waiting boundary data put into an array to send data to them later in the code. Then, each movement instance in the movement list is executed and necessary movements and changes in the sub-grid are made.

In the move method of air class, a helper method calculates the number of units that an air unit can attack for each adjacent cell. In this helper method, whether there is an unit of another faction in adjacent cells and skipped neutral adjacent cells is checked as mentioned in the project description. If one is found in adjacent cells, the number of units is incremented by 1. The neighboring sub-grids are also checked. After movement phase, the update sub-grid data is synchronized between the workers and attack phase starts.

In the attack phase, the decision of attacking is made, and if a unit decides to attack, its attack instance is added to a list. Then, according to the positions of the attacked units, the neighbors that are waiting attack instance data are found and added to a list. Then, in the resolution phase every attack instance is executed and damages applied to the units. The total damage given to each earth unit is firstly summed up, then at the end of phase, it is applied to the earth units with damage reduction effect, as mentioned in the project description. After that, the data are synchronized and the attack power of fire units increased if they killed an unit in that round. To implement this, firstly every attack instance is executed again. If a fire unit and the attacked unit are in the same sub-grid and the attacked unit is died, The attack power of attacking fire unit is increased directly. Otherwise, it sends the attacks instance to the corresponding neighbor. Then, the workers waiting attack data list is iterated and receive necessary attack instance from the sender. It also checks if the attacked unit is died that round or not. Then finds in which neighboring worker the attacker is in, and adds this worker into a list. After that, this list is sent to each neighbor and iterated, then the attack power of that fire unit is increased.

At the end of a round, the total damage taken of earth units is reset to 0 and the data are synchronized. At the end of a wave, the attack powers of fire units are reset to base value and flood operations of water units are executed. In flooding, the same logic is applied. The adjacent cells are checked and possible flood positions are added to a list. Then, by looping that list, the water units are added to the grid in calculated positions. Then, the sub-grids of each worker are sent to the manager.

4 Partitioning Strategy

4.1 Strategy Used

Specify which partitioning strategy was used (Striped or Checkered) and justify your choice.

I used checkered partitioning because initially I thought that striped partitioning might be too easy and I challenged myself, but now I think that striped partitioning might not be that easy since implementing communication between processes is a struggling concept regardless of the partitioning strategy.

4.2 Communication Between Processes

Explain how data is communicated between processes (e.g., boundary communication in striped/checkered partitioning).

Firstly, the current sub-grid of a worker is sent to all neighbors of that worker. In the forum, it is said that we shouldn't send the entire sub-grid of a worker, but it is too late for me to change this implementation. When a worker needs to check a cell in the neighboring sub-grid, it takes necessary data, but does not make operations on them. When the decision of a worker affects the neighboring sub-grids, that worker sends the data to the corresponding neighbor, and the neighbor that is taking this data does the necessary operation. For example, when an air unit in the sub-grid of worker 4 tries to move to the sub-grid of worker 2. The worker 4 computes everything, and sends the position and the object instance of the moving air unit to the neighboring worker 2. The neighboring worker 2 places the air unit in that position and the worker 4 removes the air unit in its sub-grid. When sending data to the neighbors, it is guaranteed that the number of sends are equal to number of receives because the workers that are waiting data are added to a list. Otherwise, many deadlocks can occur. Also, sometimes some workers receives data from any source because it is guaranteed that only one worker send data to that worker.

4.3 Advantages and Disadvantages

Discuss the benefits and potential downsides of your chosen strategy in terms of workload balance, communication cost, and ease of implementation.

In checkered partitioning, if the data in the grid is non-uniform, then the load will be distributed more evenly with respect to striped partitioning. This means better resource utilization since workers are rarely idle.

The communication cost in checkered partitioning is higher than that of striped partitioning since a worker have more neighbors in checkered partitioning. In striped partitioning, a worker only needs to communicate with its immediate neighbors (above and below). In checkered partitioning, a worker may have up to 8 neighbors. This can increase the complexity and overhead of communication.

Striped partitioning is easier to implement since it only needs to exchange data with the neighbors above and below. But, implementing communication with neighbors in checkered partitioning is more complex since it needs the careful computation of indices of sub-grids. Also, each partition exchanges data with up to 8 neighbors, which means more complex logic for boundary conditions.

5 Test Results

Provide the results of your tests. Include examples of initial and final grid states and discuss performance metrics such as execution time and communication overhead.

5.1 Example Input and Output

Include an example of the input and the corresponding output for a test case.

Input: my_input1

```
8 2 2 4
Wave 1:
E: 0 0, 1 1
F: 2 2, 3 3
W: 4 4, 4 5
A: 4 6, 7 7
Wave 2:
E: 1 0, 2 1
F: 3 2, 4 3
W: 5 4, 6 5
A: 7 6, 0 7
```

Output: my_output1

```
Wave 1 (Initial State)
E . . . . . . .
. E . . . . . .
. . F . . . . .
. . . F . . . .
. . . . W W A .
. . . . . . . .
. . . . . . . .
. . . . . . . A

Wave 1 (Final State)
E . . . . . . .
. E . . . . . .
. . F . . . . .
. . . . W A . .
. . . . . W . .
. . . . . A . .
. . . . . . . .
. . . . . . . .
```

```
Wave 2 (Initial State)
E . . . . . . A
E E . . . . . .
. E F . . . . .
. . F . W A . .
. . . F . W . .
. . . . W A . .
. . . . . W . .
. . . . . . A .

Wave 2 (Final State)
E . . . . . . .
E E . . . . . .
. E . A . . . .
. . F . . . . .
. . . . . . . .
. . . . W . A .
. . . . . W . .
. . . . . . . .
```

Input: my_input2

```
12 3 3 3
Wave 1:
E: 2 9, 11 7, 4 3
F: 10 10, 5 2, 6 11
W: 0 4, 7 0, 1 8
A: 11 2, 3 6, 5 9
Wave 2:
E: 8 1, 10 3, 2 5
F: 1 4, 0 11, 7 10
W: 11 5, 9 6, 2 0
A: 3 3, 4 7, 10 1
Wave 3:
E: 6 2, 9 10, 8 11
F: 4 9, 0 8, 5 1
W: 6 7, 1 1, 3 0
A: 2 2, 10 4, 7 3
```

Output: my_output2

```
Wave 1 (Initial State)
. . . . W . . . . . .
. . . . . . . . W . . .
. . . . . . . . . E . .
. . . . . . A . . . . .
. . . E . . . . . . . .
. . F . . . . . . A . .
. . . . . . . . . . . F
W . . . . . . . . . . .
. . . . . . . . . . . .
. . . . . . . . . . . .
. . . . . . . . . . F .
. . A . . . . E . . . .

Wave 1 (Final State)
. . . W W . . W . . . .
. . . . . . . . W . . .
. . . . . . . A . E . .
. . . . . . . . . . . .
. . . E . . . . . A . .
. . F . . . . . . . . .
W . . . . . . . . . . F
W . . . . . . . . . . .
. . . . . . . . . . . .
. . . . . . . . . . . .
. . . . . . . . . . F .
. . A . . . . E . . . .

Wave 2 (Initial State)
. . . W W . . W . . . F
. . . . F . . . W . . .
W . . . . E . A . E . .
. . . A . . . . . . . .
. . . E . . . A . A . .
. . F . . . . . . . . .
W . . . . . . . . . . F
W . . . . . . . . . F .
. E . . . . . . . . . .
. . . . . . W . . . . .
. A . E . . . . . F .
. . A . . W . E . . . .
```

Wave 2 (Final State)

```
. . W W W W W W W . . F
W . . . F . . . W . . .
W . . . . E . . . E . .
. . . . . . . . . . . .
. . . E . . . A . A . .
W . F . . . . . . . . .
W W . . . . . . . . . F
W . . . . . . . . . F .
. E . . . W . . . . . .
. . . . . . W . . . . .
. A . E W . . . . . F .
. . . . A W . E . . . .
```

Wave 3 (Initial State)

```
. . W W W W W W W . . F
W W . . F . . . W . . .
W . A . . E . . . E . .
W . . . . . . . . . . .
. . . E . . . A . A . .
W F F . . . . . . . . .
W W E . . . . W . . . F
W . . A . . . . . . F .
. E . . . W . . . . . E
. . . . . . W . . . E .
. A . E W . . . . . F .
. . . . A W . E . . . .
```

Wave 3 (Final State)

```
W W W W W W W W W W . F
W W . . W W W . W . . .
W W . . . E . . . E . .
W . . . . . . . . . . .
W . . E . . . A . A . .
W F . . . . W . . . . .
W W E A . . . W . . . F
W . . . W . . . . . F .
. E . . . W W . . . . E
. . . W . . W . . . E .
. A . E W W . . . . F .
. . . . A W . E . . . .
```

5.2 Performance Analysis

Discuss the performance of your implementation. Include any benchmarks or comparisons made.

I ran my code with your input and my two custom inputs with varying number of workers. Here are the results:

P - 1	input1	my_input1	my_input2
1	0.0030105	0.0035016	0.0049989
4	0.0065000	0.0059952	0.0113625
9	0.0151216	0.0178837	0.0252444
16	N/A	N/A	3.1956658

Table 1: Running times of inputs with different number of workers (in seconds)

It can be seen that, as the number of workers increases, the running time of the program increases. This may be because of communication overheads or unbalancing of loads even in checkered partitioning. Also, as the number of units or the size of the grid increases, the running time is also increases which is expected (my_input2 is 12*12 and has 9 rounds with 12 units each wave).

6 Conclusion

Summarize the overall experience of working on this project. Discuss lessons learned, potential improvements, and the benefits of using MPI for this type of simulation.

Working on this project was different than that of our prior projects, because it contains parallel execution. It was our first parallel programming project. The most challenging part of this project was implementing communication between threads. I faced deadlock situations most of the time. Debugging and fixing them took too much time. Also, this project may be the project I spent the most time on. However, even though this project struggled me very much, there were entertaining times while implementing it. Also, I learned the concepts process/thread communication and synchronizing and added MPI to my tech-stack. Without MPI, sending and receiving messages might be too hard. Also, mpi4py is easy to use, streamlining the development of the project.

Appendices

The Google Drive link for input and output files:
Click Here

References

List any references or resources you used while completing the project.

- MPI Documentation: `https://mpi-forum.org`
- Python mpi4py Documentation: `https://mpi4py.readthedocs.io`