CMPE 300 - MPI Programming Project Report

Cengiz Bilal Sarı
2021400201

Mete Damar
2021400213

December 20, 2024

1 Introduction

This report presents the implementation of a parallel simulation of an elemental battle using the Message Passing Interface (MPI) framework. The simulation models interactions between four factions—Earth, Fire, Water, and Air—on an $N \times N$ grid. The objective is to evaluate and simulate faction interactions in a distributed computing environment using efficient grid partitioning and inter-process communication strategies.The assumptions included in the project and strategically crucial points are explained in detail.

2 Design Decisions and Assumptions

The simulation adheres to the following design principles:

2.1 Design Decisions about Structure Of Units and Grid

- To create more modular and compact project, the unit class is created and the unit types are inherited from this class. A base class `Unit` defines shared attributes and behaviors, such as health, attack power, healing rate, and attack patterns. Each specific unit type (e.g., `Earth`, `Fire`, `Water`, `Air`) inherits from the `Unit` base class and implements unique attributes or overrides methods to reflect faction-specific mechanics.

- The battlefield is represented as a grid, implemented using a `Grid` class that manages the placement and state of units.

2.2 Design Decisions about Implementation

- **The most important decision:**
  Since we started the project at the first day of the project, we thought that the rows should not be directly given between processors and the desired thing is given just indices and looking these indices, so we tried hard to implement that since boundary communication and synchronization are much more hard with that. So we do not give the any rows directly and we do boundary communication between neighbour processors with required coordinates whenever there is a unit in boundary other than neutral ones. We believe that this implementation decision should be crucial for you. (This part is explained in Communication issues(3.3) part in more details.) (Also this part is explained to our assistant by Cengiz Bilal Sarı.)

- Waves and rounds are processed in parallel in each worker processor, with movement, action, resolution, and healing phases executed in order.

- The helper functions and boundary functions are written in their class since they are not directly used methods in the main class functions like phases etc.

- The MPI communication is created to communicate between workers in each round and with manager in each wave. The phase functions are written in main class and the main function basically use them in three if else block according to whether the process number is even or odd and also whether if manager or worker.

2.3 Assumptions

- Unit actions prioritize healing if their health falls below 50%.

- The fire attack power increases if it the enemy unit can be destroyed by just its attack.(enemy health- attack power of fire<= 0)
- All processes synchronize at each phase to ensure consistent state transitions with sending signals to each other .
- If two Air units tries to move to a same coordinate, they merge. That means their attack power and health are summed up but health can't exceed the full health.
- The Air unit do not calculate the move possibility of other air units while calculating the attack number in windrush special ability.
- The special ability of faction Earth reduces incoming damage by 50% (rounded down). The attacks are reduced by half for each attack and we put the halved damage to attacks queue.
- The air units could pass just one neutral unit.
- The number of processors and grid size are given such that grid can be divided into worker processors with equal sizes.

3 Implementation Details

3.1 Implementation of classes and main MPI structure

The first thing we made is creating appropriate classes for project and partition between worker processors. After writing the unit and grid classes, the stripe partitioning method is written for partitioning and the grid is divided to worker processors. The synchronization is achieved by comm.barrier after each partitioning in manager processor and start of the wave in each worker processor.

3.2 Deadlock issues

We examined that the deadlock happens a lot since the worker processors want to receive data from each other at the same time and waiting each other indefinitely. So we use odd and even structure you suggest to avoid from deadlocks. The phases are done like this:

- In phases where communication is needed, first even processors do their phase and odd number processors provide information to evens.After evens are done with their phase, they send signal to odds to indicate the end of the phase. After all evens are done the odds are start to their phase and evens provide information.
- While providing to other processors, also provider obtain information about where will be attacked in their grid part in action phase, or whether new air comes to them or not in movement phase .

3.3 Communication Issues and Details

The communication is created between the neighbor processors whenever it is needed. The rows are not exchanged directly just the necessary coordinates are exchanged between processors. To be more descriptive the case given:

. . . E . . . . (P1 boundary)
. . . . W . . . . . A . . . E . (P2 boundary)

In this case the P2 sends the necessary coordinates which P1 should look and the attack power of the attacker to P1. So, for water unit just the diagonal indices will send to P1 and P1 look at these indices. Then if there is a unit which can be attacked, the coordinates are put into attacks queue of P1 and the signal send to P2. P2 takes the information whether the unit can attack or not for healing phase.
For the air unit we also send how many rows it should go ahead, for example if this information is 2, the P1 goes 2 times in each direction until it cross an unit other than neutral, then the other parts are the same with above explanation.
This implementation part was so hard for us to calculate edge cases etc. We tried to make the project in a suitable way to logic of parallel structure and create boundary communication whenever needed with the least size of exchanged data can be achieved.

3.4 Air Movement Details

For the air movement part, the air simulation is made like it goes to the adjacent coordinates. Then the air does action phase in simulation which

actually does not attack and calculates the number of possible attack. This action simulation is just made like in action phase and communication is the same.
The passing neutral unit is a hard part of this phase in boundary communication part. The another specific case is given about boundary communication:

```
. . . E . . . . . . . . . . . . (P1)
. . . . . . . . . . . . . . . . (P1 boundary )
. . . . . . . . . . . . . . . . (P2 boundary)
. . . . W . . . . . A . . . E . (P2 )
```

Here if the air moves to upper row, it can move 2 rows in P1 part, so with the directions also this number is sended to P1, then P1 looks at the directions with passing the neutrals until the 2 rows which it should look at ended.

4 Partitioning Strategy

4.1 Strategy Used

The implementation uses a striped partitioning strategy. Each worker process is assigned a contiguous subset of rows from the grid. Key considerations include:

- Equal division of workload among processes in terms of grid part that assigned to worker processors.
- Reduced complexity in boundary communication compared to checkered partitioning.
- Efficient inter-process data exchange to optimize memory usage.

4.2 Boundary Communication

Important note: Since the details and decisions about boundary communication is explained in more detail at above parts (2.2, 3.3), we explained other parts about boundary communication here which we did not mention above.

- Neighboring processors exchange boundary cell coordinates that needed.

- Since we get some errors about data when we give it to MPI send and receive methods, we serialize the data and give the byte array to MPI methods. Then we deserialize it in receiving processor.

  The `pickle` library serializes grid data for MPI communication. It does not change any logic and process of the project we just use it to give byte array to MPI send and receive methods.

- Recursive queries in each direction for air unit handle cases like Air unit attacks that cross partition boundaries.

4.3 Advantages and Disadvantages

4.3.1 Workload Balance

- Striped partitioning divides the grid into equal parts on worker processors, so the workload balance between workers in average is approximately the same. The workload can change case by case since the unit numbers other than neutral ones could change, so the communication and calculation complexities may alter. But in average it should be more or less the same since the waves and rounds are arranged randomly.

- In another project, the partitioning can be done in a way that also provide the complexities and calculations are divided into processors dynamically with more equal concept, but it might increase the complexity and overhead of the communication.

4.3.2 Communication Cost

- Since we do not give the 3 boundary rows at the start of each round, the unnecessary data exchanges are not created. The data size we send and receive between processors is much less since we just send the coordinates required. But it might also increase the overhead at overall in terms of time if there are so much units at the boundaries.

- The communication might be reduced especially in movement phase, since we need to find the number of possible attack for 9 coordinates, the finding this number at boundaries might be done just with one communication for movement and calculate all of them at once which create another complexity for the implementation.

4.3.3 Ease of the Implementation

- The striped partitioning provide easier implementation on behalf of deadlock issues compared to checkered partitioning. The deadlock could be solved by odd-even scheduling in phases which might not enough for checkered partitioning.

- The strategy in boundary communications we made create complexity for us but it is more suitable for the logic of this project. It could be optimized with doing boundary communication in two way according to number of units that are in the boundary.(If it is more than some threshold, give the rows directly etc.)

5 Test Results

5.1 Example Input and Output

We provide grid of the current healths for just one basic and one hard test case example to see the structure better.

Input: Basic case to see generic correctness

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

Output:

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


.  .  .  .  .  .  .  10
18  .  .  .  .  .  .  .
.  18  .  .  .  .  .  .
.  .  12  .  .  .  .  .
.  .  .  6  .  .  .  .
.  .  .  .  14  .  .  .
.  .  .  .  .  10  .  .
.  .  .  .  .  .  .  .
```

Input:

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

Output:

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

Input:

```
8 1 2 3
Wave 1:
E: 1 3, 0 3
```

```
F: 7 7, 0 5
W: 1 1, 7 0
A: 7 1, 3 3
```

Output :

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

Input:

```
16 2 2 10
Wave 1:
E: 13 5, 15 4
F: 11 5, 9 9
W: 11 2, 6 9
A: 9 14, 5 0
Wave 2:
E: 14 3, 7 2
F: 12 11, 13 8
W: 15 11, 14 8
A: 12 7, 12 1
```

Output:

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

Input: More complex example with air movements and multiple waves

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

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

18  .  .  .  .  .  .  .
18  18  .  .  .  .  .  .
.  7  3  6  14  .  .  .
.  .  10  .  4  .  .  .
.  .  .  2  6  .  .  .
.  .  .  .  4  .  7  .
.  .  .  .  .  4  .  .
.  .  .  .  .  .  6  .
```

Input:

```
24 2 4 4
Wave 1:
E: 23 16, 10 10, 23 20, 11 16
F: 23 6, 9 8, 1 2, 23 19
W: 11 23, 20 0, 5 10, 23 10
A: 14 1, 19 2, 2 21, 6 22

Wave 2:
E: 16 15, 2 5, 18 0, 0 11
F: 16 4, 6 3, 15 7, 7 8
W: 3 1, 5 21, 10 17, 14 5
A: 9 6, 13 0, 12 14, 0 6
```

Output:

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

5.2 Performance Analysis

5.2.1 Theoretical Analysis

Let the following variables represent:

- N: Grid length (grid size is $N \times N$)
- P: Number of worker processors
- W: Number of waves
- R: Number of rounds

For the interconnection networks, general rule for the basic operation is :

- Parallel assignment statements + Work within processors (rare)
- Time complexity $\approx$ Communication complexity

First we could examine the most effective sequential parts inside of the processors and then the communication parts which is more crucial. We will analyze the worst-case scenario where all active fractions lie on the grid border which corresponds to the most communication.

The most effective sequential parts inside of the processors:

- For all the phases namely movement, action, resolution and healing, The time consuming part basically stems from "iteration through grid partitions" and it happens for each round and wave so :

  W(n)= $W * R * \frac{N^2}{P} * k \in \theta\left(W * R * \frac{N^2}{P}\right)$,where k is some constant

The communication complexity in the worst case:

- Parsing and Broadcasting: The manager process broadcasts the data to all worker processes.
  - Broadcasting complexity: $\theta(P)$
- Movement and Action Phases
  In the calculation of the air movement part, if the boundaries are all air, it is the worst case scenario. For the action phases the most communication could acquire if the boundaries are full of units which are not neutral.
  - Boundary communication complexity: $\theta(N)$
- Flood Check
  For boundary water units, inter-process communication occurs, so in the worst case for full of water units at the boundaries:
  - Boundary complexity: $\theta(N)$

Total complexity of the boundary communication:
These phases occurs for each round, wave, the broadcasting occurs for each wave, then in total the communication complexity :

- $W(n) \in \theta(W * (P + R * N))$

5.2.2 Evaluation of the Strategy and Some Informations From Run Time

For the run time analysis , we changed 2 criteria one by one with keeping the other the same for each try:

- The size of the grid
- The intensity of the units other than neutrals (especially in boundaries)

Evaluation of the parameters:

- The size of the grid did not affect the communication so much compare to complexity of the communication while we are keeping other parameters the same. The intensity of the units especially in boundaries affects the complexity much more since it is the bottleneck of the parallel algorithms.

Evaluation of the strategy:

- In striped partitioning , inter-process communications can occur only in borders. Increase in factions in the borders will increase the number of inter-process communications. Since there is N column, the inter-process communications will have a complexity of $\theta\left(N\right)$ for each round and each wave and also $\theta(P)$ for each wave comes from broadcasting , and as we calculated, the complexity of inner-process operations is $O\left(\frac{N^2}{P}\right)$ for each round and wave. We know that inter-process communications are much more slow than inner-process operations. Therefore, there are a few important points to consider:

- As explained in the report, the boundary communication are not done in advance of the each round in our code. It is done dynamically whenever it is needed in boundaries. So, if the ratio of elements in the boundaries and the size of the row in the grid (N) , we need to do much more communication compare to taking them in advance so it might create overhead. But for the logic and efficiency of the parallel programming the boundary communications need to be done whenever it is needed with the least data it could be done.This is important because when density of border factions are low, other implementations will send and receive again the whole border rows.

6 Conclusion

The project demonstrates the effective use of MPI for simulating complex interactions in a distributed computing environment.With the challenges faced about deadlock issues and synchronization, we gained the instinct about the hardness of the synchronization of parallel programming without creating any undesired behaviour.

Striped partitioning provide efficient parallel programming for the given requirements, though additional strategies such as checkered partitioning could enhance scalability and balance in diverse scenarios. The mechanisms we create for the feature of the Air and Water units could be optimized in the implementation to reduce the effect of overheads. The code structure we provide supports modularity and extensibility of the project for future improvements.

Appendices

—

References

- MPI Documentation: `https://mpi-forum.org`
- Serialization of the Data: `https://www.datacamp.com/tutorial/pickle-python-tutorial`
- Python mpi4py Documentation: `https://mpi4py.readthedocs.io`