CmpE 300 Analysis of Algorithms
Project-2 Report

Halil Burak Pala - 2019400282
Berat Damar - 2018400039

January 2022

1 Introduction

We have completed the project and our code is running for the given testcases. We preferred Python programming language and we did not use numpy library.

2 Structure of Implementation

In our project, we used striped structure.

In the manager process, before sending map parts to the worker processes, we first read the input file, and initialized the map according to the first lines of o and + castle locations. Then we get the coordinates for each wave, put them into different lists for o and + castles. After these, we sent all these coordinates and corresponding map parts for each process to the processes separately. This is the tag-1 communication.

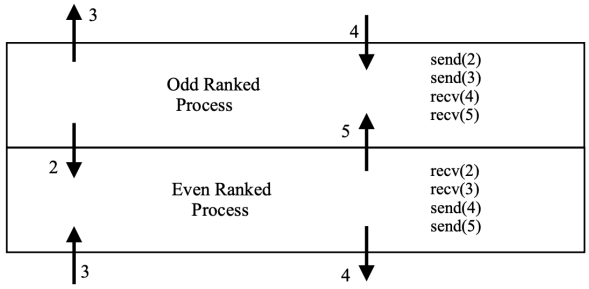
We sent coordinates for all the waves from manager to all workers at once, and then during the initialization of the processes, we extracted the coordinates that corresponding process is responsible from, which is the coordinates that are included in the map part given to that process. By that way, during the update of the map parts, all processes updated corresponding map parts and did not interfere with other parts of the map.

After the initialization of processes, we simulated the game. During the simulation, if a neighboring elements does not exist, this neighboring element is assumed as a dot. Necessary arrangements for this assumption are done in the code. Other parts of the simulation are done as in description.

Every process should receive its neighbors' boundary rows and also every process should send its boundary rows to its neighbors. This communication between worker processes is done as follows:

After a round of the game finishes, odd ranked processes first send their upper rows and lower rows to their neighboring even ranked processes. After even ranked processes obtain these rows, they also send their upper and lower rows to their neighboring odd ranked processes. The tag-2, tag-3, tag-4 and tag-5 communication channels are used for these communications.

For the first and last rows, special considerations are necessary. For the first process, since it has no process above it, it does not sent a row to or receive a row from a process above it. So, it does not have tag-3 and tag-4 communications. For the last process since it has no process below it, it does not sent a row to or receive a row from a process below it. But there is two possibility for the last process: If it is an odd ranked process, it does not have tag-2 and tag-5 communications. If it is an even ranked process, it does not have tag-3 and tag-4 communications. An illustration for the tags is given below:



After the game finishes, the processes should send their map parts back to the manager process. In order to avoid interference, we implemented this part as sequential, i.e. beginning from process 1 to the last process, every process sends their map parts in order. These are done by communication channels with tags 5+rank. Manager process concatenates these map parts, and then prints the final result to the output file. In essence, our final communication scheme is as follows:

- **Tag-1:** Manager process sends packets necessary for workers to them.
- **Tag-2:** Odd ranked processes send their lower row to the process below them. Even ranked processes receive these rows.
- **Tag-3:** Odd ranked processes send their upper row to the process above them. Even ranked processes receive these rows.

- **Tag-4:** Even ranked processes send their lower row to the process below them. Odd ranked processes receive these rows.
- **Tag-5:** Even ranked processes send their upper row to the process above them. Odd ranked processes receive this row.
- **Tag-(5+process rank):** At the end of the simulation, every process sends its map parts back to manager process.

3 Analysis of the Implementation

3.1 Analysis in terms of communication between processes

In communication analysis, we assumed sending one item, such as one integer value in a list, takes $O(1)$ time.

- **From the manager to workers**, P number of serial communications take place. During each of these communications we send the following:

  (1) Map part that the corresponding process is responsible from

  (2) Coordinates in each wave that the corresponding process is responsible from

  Here, (1) is consisted of $\frac{\sqrt{n}}{P}$ rows and each of these rows is consisted of $\sqrt{n}$ elements, so a total of $\frac{n}{P}$ elements are sent.

  (2) has 2 lists of lists in it, every list of lists has $W-1$ lists in it. Here, we assumed that on average, every process is responsible from $\frac{T}{P}$ number of coordinates while updating its map part in each wave. So, every list has $\frac{T}{P}$ pairs. Every pair consists 2 element. Hence, a total of $2 \cdot (W-1) \cdot \frac{T}{P} \cdot 2 = 4 \cdot (W-1) \cdot \frac{T}{P}$ elements are sent while sending (2).

  Since P number of serial communications take place, this part takes a total of $= O((\frac{n}{P} + 4 \cdot (W-1) \cdot \frac{T}{P}) \cdot P) = O(n + WT)$ time on average.

- **From the workers to the manager**, P number of sequential communication take place. We send map parts to the manager process and every map part consists of $\frac{n}{P}$ items. So, in terms of time complexity, this part takes $O(\frac{n}{P} \cdot P) = O(n)$ time on average.

- **Between workers**, $\frac{P}{2}$ number of parallel communications take place because of the communication scheme that we discussed. In each round, 4 serial communications take place. During these communications, one row of data which consists of $\sqrt{n}$ items are sent. Since there are W number of

waves and every wave is consisted of 8 rounds, a total of $4 \cdot 8 \cdot W = 32W$ communications take place. So, in terms of time complexity, this part takes $O(32W \cdot \sqrt{n}) = O(W\sqrt{n})$ time on average.

Here, if we had serial communication between workers, we would have $O(P \cdot W \cdot \sqrt{n})$ time complexity. Here, by the help of parallel communication between processes, we speed up the implementation. But we have to send every process its corresponding data from the manager at first. This operation takes $O(n + WT)$ time in the implementation and so, slows down the implementation.

3.2 Analysis of the work within processes

(Here, we analyzed the work done within the processes. While doing these analyzes, we didn't go into deeper. It is a rough analysis.)

In the manager process, we read the input file, get the coordinates for each wave, initialize the map, split the map for every process and print the result to the output file. If we investigate these operations,

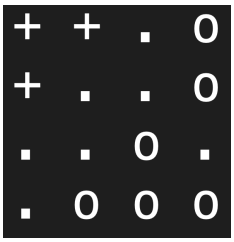
- Reading the input file includes $3 + 2 * W * T * 2 = 3 + 4WT$ operations. So, it takes $O(WT)$ time.
- While initializing the map, we do $n + 2 * W * 3 = n + 6W$ operations. So, it takes $O(n + W)$ time.
- While splitting the map for each process, we do $2 \cdot W \cdot T$ operations. This takes $O(WT)$ time.
- Outputting the results takes $O(n)$ time.

So, manager process takes $O(n + WT)$ time.

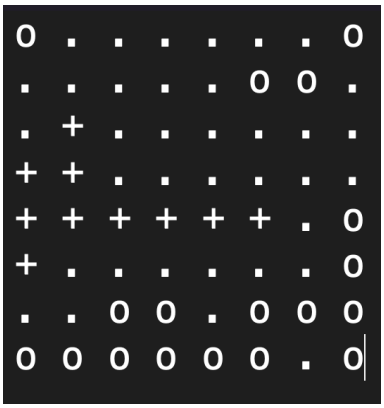
If we do an analysis of the work done in a worker process, we can say the main work done in worker processes is simulating the game. In the simulation of the game, during one round, we do $O(\frac{n}{P})$ operations. Since there are 8 rounds in one wave, and there are W number of waves, simulation takes $O(\frac{nW}{P})$ time. So, each manager process takes $O(\frac{nW}{P})$ time.

4 Test Outputs

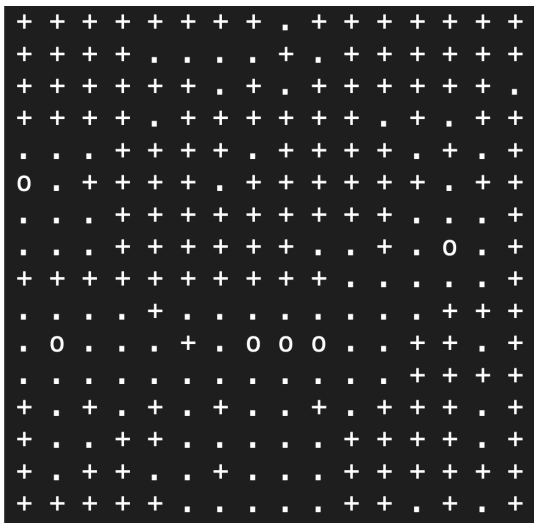
- test_output1.txt:



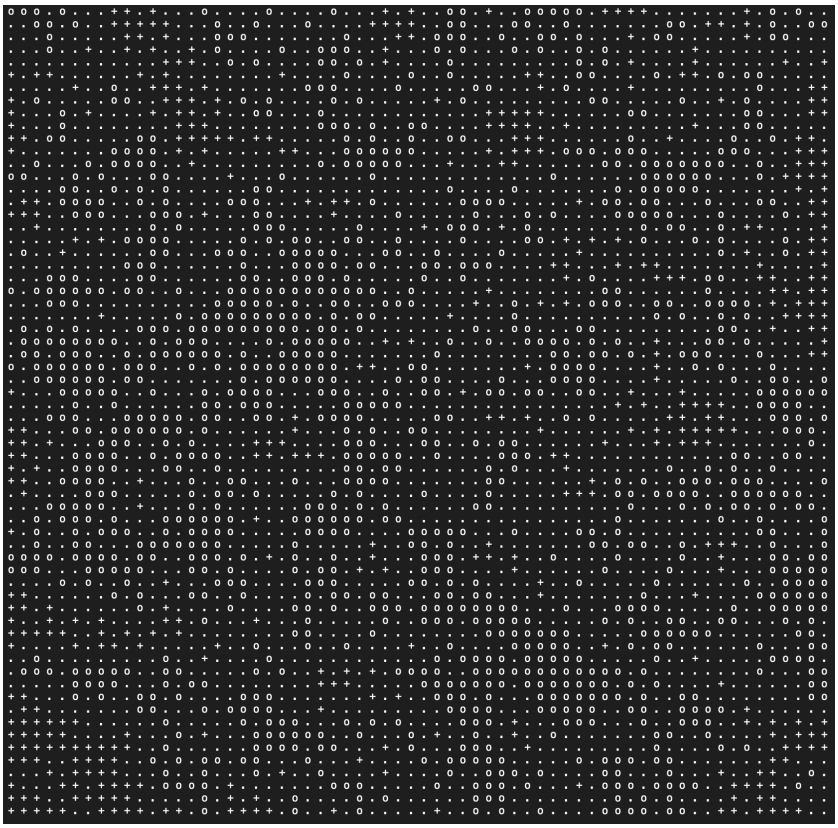
- test_output2.txt:



- test_output3.txt:



- test_output4.txt:



5 Difficulties Encountered and Conclusion

While coding, we did not encounter a significant difficulty.

In conclusion, this project provided us a good opportunity for experiencing parallel programming that we learned in the lectures. Setting up a communication structure between processes and then analyzing this structure was helpful in understanding the idea behind parallel programming.