Furkan Özdemir – 2018400201
Karahan Sarıtaş - 2018400174

CMPE300 - Analysis of Algorithms

Fall 2021

MPI Programming Project

(A) Introduction

We completed the MPI programming project. Our implementation is checkered version of the project. The code is running successfully for an even number of processors *P* that is a perfect square, that is $P = x^2$, as stated in the project description.

(B) Structure of the Implementation

In our implementation we first decided that processor with rank 0 is manager processor and the rest are worker processors. Then, manager processor first reads the input file. It sends number of *rows*, number of *waves* and number of *cells per worker* to worker processors via send() function. Then for each wave, it creates a 2D array representing a board with new added towers. It uses 0 for '.', 1 for 'o' towers and 2 for '+' towers. It sends the boards to each worker processor. At the end of the simulation, it receives the towers from worker processors and builds the final board from scratch as a 2D array. Then it opens the output file and write the board onto it.

For worker processors, each of them has two 2D integer array named *towers* and *healths*. *Towers* array keeps the coordinates of towers, while *healths* array keeps the remaining healths of the towers. Each worker processor receives number of *rows*, number of *waves* and number of *cells per worker* at the very beginning of the simulation. Then in each wave, they receive a 2D board representing the newly added towers. If there is

already a tower in the corresponding coordinate, they just ignore the newly added tower, otherwise they update the *towers* and therefore *healths* arrays.

To provide communication between worker processors, we first interchange information between rows, secondly interchange information between columns, thirdly interchange information from left below corner to right above corner and finally interchange information from right below corner to left above corner. To avoid deadlocks, first worker processors whose cells are in the odd numbered rows send their row information to other worker processors. Then, worker processors whose cells are in even numbered rows send their row information to other processors. Similarly, we applied the odd-even order for other interchange operations as described in the documentation.

Finally, we used the *update()* function, to update the information according to game rules for each eight round in *W* waves. Each worker processor traverse the cells they are responsible for, identifies neighbors and call the *update()* function with necessary arguments in order to update *healths* array.

At the end of the simulation, each worker processor sends their *towers* array to the manager processor. Manager processor builds the final board using *towers* arrays and writes the output to the output file.

(C) Analysis of the Implementation

Let's start out analyzing by defining the input variables. Let *P* denote number of worker processors, *N* denote number of rows, *W* denote number of waves and *T* denote the number of towers newly added at the beginning of each wave.

First analyze the time complexity of the operations performed by manager processor. Within the manager processor, we first send integer variables such as *N*, *W* and *cell per worker* to each processor, which takes O(*P*) time complexity since we have to traverse through the worker processors. Then in each wave, it creates a 2D board to represent newly added towers, splits this board and sends each part to the worker processors. In this series of work, bottleneck is the communication between worker processors. Overall, manager sends N^2 cells to the worker processors in each wave, therefore this step takes O($W \cdot N^2$) time complexity. At the end, we get back *towers* array of each worker and build the final board using these pieces. This

operation takes O(N^2) time complexity since we do N^2 assignment operations to build the final board. Then worker writes the output to the output file again in O(N^2) time complexity.

For each wave, worker processors take some part of the board that corresponds to the coordinates they are responsible for. This operation takes O($\frac{N^2}{P}$) time complexity since each worker is responsible for $\frac{N^2}{P}$ number of cells. Then for each round, workers communicate with each other. This is the part where we analyze the time complexity of communication between worker processors. For the worst case, let's think about the processor which has 8 neighbors. If it's labeled as odd, it first sends and then receives; if it's labeled as even, it first receives and then sends. It sends all its upper row to above, all its lower row to below, all its last column to right and all its first column to left. Each of them consists of $\frac{N}{P}$ cells. Then it sends its corners to left above, right above, left below and right below. Overall this communication takes $\left(\frac{N}{P} + \frac{N}{P} + \frac{N}{P} + \frac{N}{P} + 1 + 1 + 1 + 1\right) \cdot 2$ cell exchange including send/receive. In total it makes O($\frac{N}{P}$) time complexity. If we denote number of cells as *n* as in the description, then time complexity for communication is O($\frac{\sqrt{n}}{P}$). Observe that for a striped-split implementation, even-odd approach would give O($\sqrt{n}$). Therefore checkered-split is way better than striped-split approach as number of worker processors increases. Since this communication takes place for each round, overall time complexity becomes O($\frac{\sqrt{n}}{P} \cdot W \cdot 8$) = O($\frac{\sqrt{n}}{P} W$). It is worth noting that This complexity represents the number of steps, not the number of total operations since work is done in parallel.

Let's analyze the work within processors. Each processor iterates through the cells it is responsible for and updates the *healths* array. A processor is responsible for $\frac{N^2}{P}$ cells. Therefore time complexity of updating the cells in a round is O($\frac{N^2}{P}$), considering each round and wave in overall it becomes O($\frac{WN^2}{P}$).

Furkan Özdemir – 2018400201
Karahan Sarıtaş - 2018400174

(D) Test Outputs

-test_input1.txt

```
+ + . o
+ . . o
. . o .
. o o o
```

-test_input2.txt

```
o . . . . . o
. . . . . o o .
. + . . . . . .
+ + . . . . . .
+ + + + + + . o
+ . . . . . o
. . o o . o o o
o o o o o o . o
```

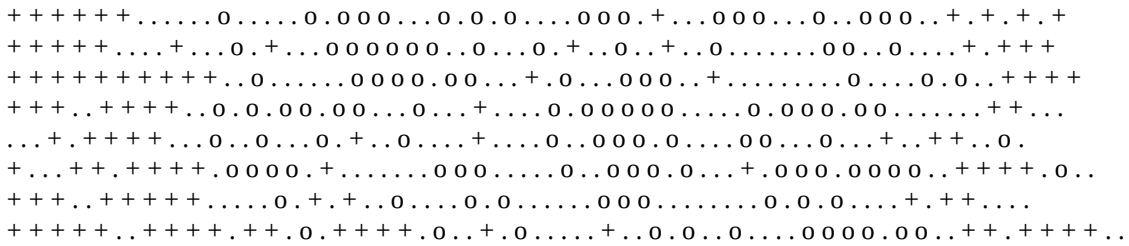
-test_input3.txt

```
+ + + + + + + + . + + + + + + +
+ + + + . . . . + . + + + + + +
+ + + + + + . + . + + + + + + .
+ + + + . + + + + + + . + . + +
. . . + + + + . + + + + . + . +
o . + + + + . + + + + + + . + +
. . . + + + + + + + + + . . . +
. . . + + + + + + . . + . o . +
+ + + + + + + + + + . . . . . +
. . . . + . . . . . . . + + +
. o . . . + . o o o . . + + . +
. . . . . . . . . . . + + + +
+ . + . + . + . . + . + + + . +
+ . . + + . . . . . + + + + . +
+ . + + . . + . . . + + + + + +
+ + + + + . . . . . + + . + . +
```

Furkan Özdemir – 2018400201
Karahan Sarıtaş - 2018400174

-test_input4.txt

Furkan Özdemir – 2018400201
Karahan Sarıtaş - 2018400174



(E) Difficulties Encountered and Conclusion

Overall, this was a satisfactory project to practice on communication in parallel programming. It is worth noting that the even-odd communication scheme for the checkered version was one of the difficulties we encountered during the implementation.