Project Report

Alp Tuna - 2019400288

Mustafa Atay – 2020400333

1) Introduction:

We have completed the project. Our code runs successfully. Only when process number is 65, we encountered a system error. This is about hardware, so we don't think that this will be a problem. We compared our outputs with the given ones and they were same. We believe that our code contains no bugs.

2) Structure of the Implementation: We implemented striped version of the project. As stated in the description, even numbered processes and odd numbered processes send and receive the data consecutively. However, manager process sends the data to other processes in one step and receive the output in one step. Hence, file i/o is done only in the beginning and in the end. We didn't make any assumptions. In the beginning of the each round processes receive the data and send their data in edge lines. Hence, we don't send any additional unnecessary information and receive don't lose any necessary data. Below is the ss of the send/receive part for odd/numbered processes. Even numbered processes are same just the order is different.

```
for round_num in range(8):
    #send above --> tower_arr[0]
    if(rank > 1):
        comm.send(tower_arr[0], dest=rank-1, tag=1)

    #send below --> tower_arr[len(tower_arr)-1]
    if(rank < p_num):
        comm.send(tower_arr[len(tower_arr)-1], dest=rank+1, tag=1)

    #receive from above
    if(rank > 1):
        data_above = comm.recv(source=rank-1, tag=1)

    #receive from below
    if(rank < p_num):
        data_below = comm.recv(source=rank+1, tag=1)
```

3) Analysis of the Implementation: As we have stated in the lectures, main overhead in the code is interprocess communication. Hence, we should consider send and receive operations as our basic operations. In each round, each process except the manager process sends data 2 times and receives data 2 times. Only processes on edges of the map do these operations once. Hence our complexity is as follows:
Number of rounds = W*8
Number of send and receive operations for usual processes: W*8*4 = 32*W (2 sends and 2 receives)
Number of send and receive operations for processes on edge: W*8*2 = 16*W (one send an done receive)

Hence, number of basic operations is linearly dependent on number of waves. So, complexity of our algorithm is:

$$\Theta(W)$$

```
for i in range(W):
```

…… Some code without loop etc.

……..

……..

```
for round_num in range(8):
                #send above --> tower_arr[0]
                if(rank > 1):
                    comm.send(tower_arr[0], dest=rank-1, tag=1)

                #send below --> tower_arr[len(tower_arr)-1]
                if(rank < p_num):
                    comm.send(tower_arr[len(tower_arr)-1], dest=rank+1, tag=1)

                #receive from above
                if(rank > 1):
                    data_above = comm.recv(source=rank-1, tag=1)

                #receive from below
                if(rank < p_num):
                    data_below = comm.recv(source=rank+1, tag=1)
```

4) Difficulties Encountered and Conclusion:

Working as a group is always a challenge. Division of the work and modularity/readability of the code etc. This was one of the difficulties we cannot avoid. Specifically for this project, learning and installing mpi was a little challenging. We couldn't set up the environment somehow and tried many things until being able to run. This was the second challenge. Since we had limited time we didn't dare to implement checkered implementation. Enormous number of nested if/else and for loops was the main challenge. We couldn't avoid it and our code even started to look like functional programming where you cannot understand anything due to nested parantheses. Hence, this was the third and most significant challenge we had. However, overall it was a nice project and we learned a lot about parallel processes. We see these subjects in other courses(such as cmpe344) as well. So it was nice to implement it in a real project!