# **Introduction**

Project is completed. When our code run with given five inputs and different number of processors, it gives correct outputs as long as number of processors is divisible to √ n

# **Structure of the Implementation**

Our code basically consists of three parts: reading and distributing data to processors, data exchange between processors / simulating game and returning final data to manager processor. Each processor has three arrays: one for their original map, two for storing data from processors above and below, however, first and last processors have empty arrays (above for first, below for last) just for sake of being parameter to simulation function. We implement our code with striped approach that we felt comfortable with. We tried to partition processors as even/odd to prevent any deadlocks. Our version of communication scheme is as follows:

Each even numbered processor (2p):

* Sends bottom row to *above* array of processor 2p + 1 (except for last processor)
* Sends top row to *below* array of processor 2p – 1
* Receives top row of processor 2p + 1 to *below* array (except for last processor)
* Receives bottom row of processor 2p-1 to *above* array

Each odd numbered processor (2p + 1):

* Sends bottom row to *above* array of processor 2p + 2
* Sends top row to *below* array of processor 2p (except for first processor)
* Receives top row of processor 2p + 2 to *below* array
* Receives bottom row of processor 2p to *above* array (except for first processor)

Like it is stated in the project description we did not encounter any deadlocks with this structure, since our code never attempts to receive or send data to same space (array) at the same time.

# **Analysis of the Implementation**

Since processors does not have to wait for receiving or sending, we must only consider number of processors for calculating time complexity of processor communications. Each processor except for the first and last one makes exactly two send and two receive operations, which means total number of operations is 4p – 2, that is, time complexity is O(p). Since p is divisible to √ n it can be said that time complexity is also equals to O(√ n).

As for codes within processors, since two for loops which iterates √ n x (√ n / p) times totally, it can be said that complexity is also O(√ n).

# **Test Outputs**

Four test output files is in the zip file we submitted.

# **Difficulties Encountered and Conclusion**

When data size is big as it is in test\_input\_4.txt and if we run our code with more processors, running time increases as well. This is because of the communication overhead between processors, it is normal to face with slower run times when we increase the number of processes too much. With relatively small imputs, number of processors does not make any significant change to running time.

At first we could not figure out how to code even/odd process and what its purpose was and we coded that part in a whole different manner, which obviously was not efficient. After discussing and brainstorming a while, we comprehended that part completely and fixed our code.