Parallel Game of Life

CMPE300 - PROGRAMMING PROJECT

Ömer Yılmaz
TUNGO GÜNGÖR | 20.12.2019

Introduction

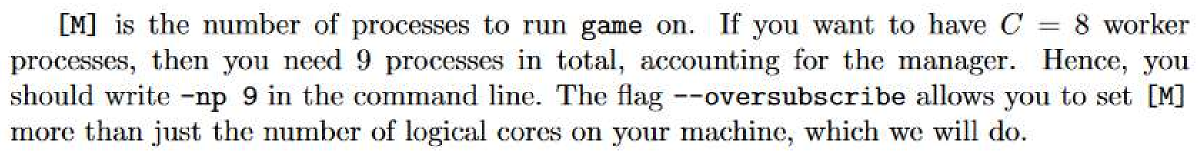
The aim of the program is simulating Conway's Game of Life on a multiprocessor system by using MPI (Message Passing Interface). We split the data to different processors and process them after that we reunite the data.

Program Interface

First of all, you need **Python3** to execute the program. After that **MPI** and **numpy** libraries should be installed. In order to install **mpi4py, "apt install libopenmpi-dev"** command must be executed. Then **"pip3 install mpi4py"** and **"pip3 install numpy"** must be executed. In order to avoid some warnings **"export OMPI_MCA_btl=self,tcp"** can be executed. After that the environment is ready. The program can be run with the following:

```
mpirun -np [M] --oversubscribe main.py input.txt output.txt [T]
```

As mentioned in the project description:



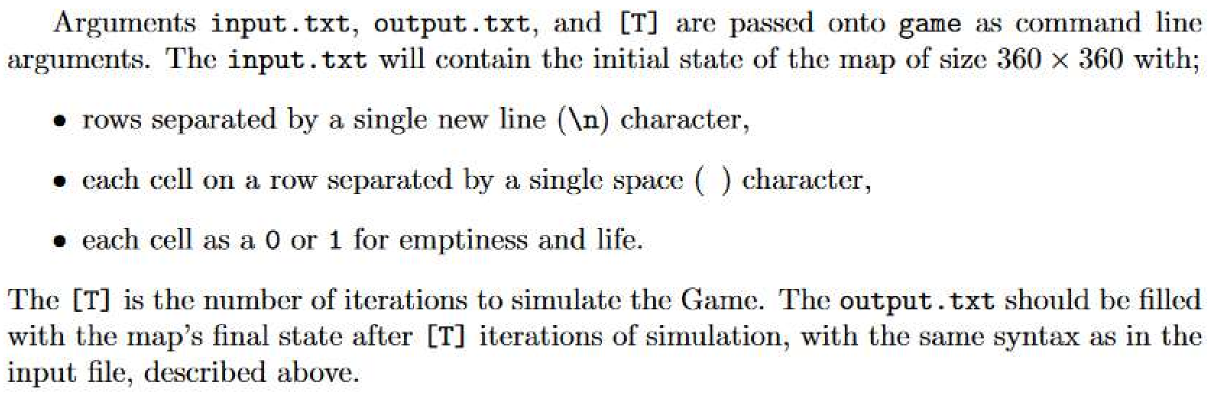
`[M]` is the number of processes to run `game` on. If you want to have $C = 8$ worker processes, then you need 9 processes in total, accounting for the manager. Hence, you should write `-np 9` in the command line. The flag `--oversubscribe` allows you to set `[M]` more than just the number of logical cores on your machine, which we will do.

Program Execution

There are not any interactions other than input and output.

Input and Output

As mentioned in the project description:

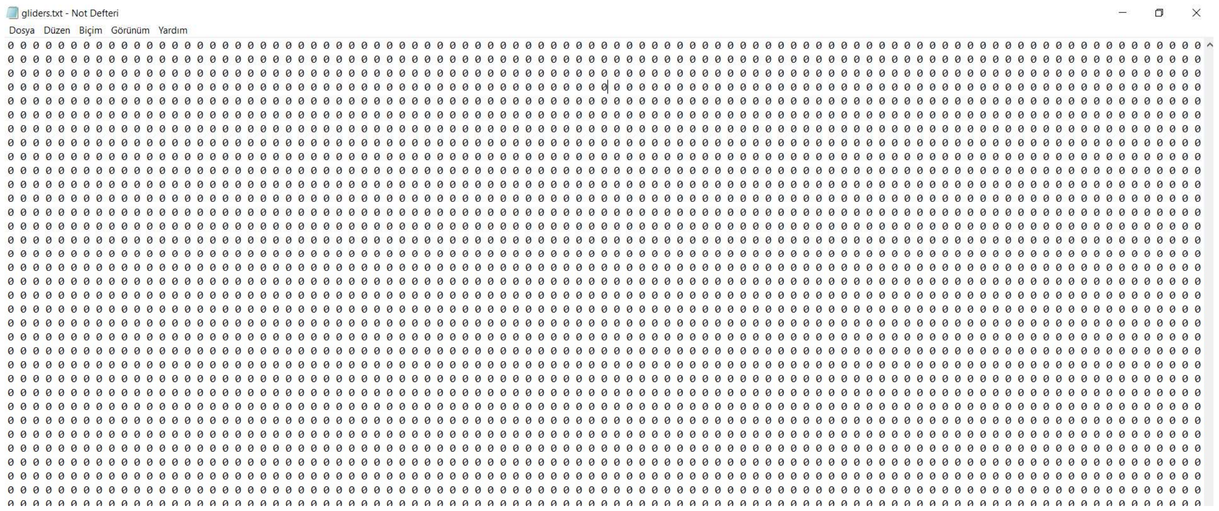


Arguments `input.txt`, `output.txt`, and `[T]` are passed onto `game` as command line arguments. The `input.txt` will contain the initial state of the map of size 360×360 with;

- rows separated by a single new line (`\n`) character,
- each cell on a row separated by a single space ( ) character,
- each cell as a `0` or `1` for emptiness and life.

The `[T]` is the number of iterations to simulate the Game. The `output.txt` should be filled with the map's final state after `[T]` iterations of simulation, with the same syntax as in the input file, described above.

Example input "gliders.txt" which is given:



```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

The output format is the same.

Program Structure

Variables:

comm: MPI structure

rank: Id of the each process

Grid_Size: Size of the input grid.

worker_count: Number of the workers

w_per_dimension: Square root of the number of the workers

Rank_size: Size of the grid of the each worker

Time: Total iteration

Input: Input file

Output: Output file

Grid: 360*360 np array of the grid

Row: Row of the worker

Col: Col of the worker

Data: Grid of the each worker

Right: The column received by the right worker

Left: The column received by the left worker

Down: The column received by the down worker

Up: The column received by the up worker

Upright: The square received by the upright worker

Upleft: The square received by the upleft worker

Downright: The square received by the downright worker

Downleft: The square received by the downleft worker

New_grid: The output after Time Iterations

Methods

To_rank(row, col): Method to get rank from column and row of the worker

From_rank(rank) Method to get row and column from the rank of the worker

Next_sqr(center, sum): Calculating the next value of a square according to game of life rules.

Next_step(upleft, up, upright, downleft, down, downright, left, right, data): Method to calculate the next situation of grid of a worker

General Structure

Lines 1-4: Necessary imports

Lines 6-14: Initializing the mpi and getting rank and getting necessary variables

Lines 16-51: Methods

Lines 54-66: If rank == 0 loads input and send the partitions to the workers

Lines 67-71: If rank != 0 then calculates the row and col and receive the data from the master

Lines 72-138: For loop for the iterations of the game.

Lines 77-84: Initializing numpy arrays that will be received from the neighbors

Lines 86-97: If the worker is a white square in chess according to its rank, it sends the necessary data after that it receives from left, right, up, down workers.

Lines 99-110: If the worker is black square it receives first, then sends the data to neighbors.

Lines 113-124: If column is even, sends the corners to neighbors then receive

Lines 125-136: If column is odd, first receive then send

Line 138: Call the next_step method to calculate the updated version of the grid

Lines 141-151: If rank == 0, initialize new grid then receive the results from the workers and prints the output.

Lines 153-154: If rank != 0, then send the local grid to the master.

Examples

As given in the moodle, gliders.txt is an example for the input file and the gliders_0**.txt are the example outputs of the program where ** are the number of iterations.

Improvements and Extensions

This program is only for the 360*360 input. If any other input is given, program will be crashed. It can be generalized to encounter this problem. Moreover, there is not any try-catch block. Therefore, if any exceptions occur the program will be crashed again. The visualization script can be included in the program in order to get a better user experience.

Difficulties Encountered

The main difficulty of the program is to get familiar with the MPI interface. I have never programmed with that interface and any other multiprocessor interface. Therefore, I was a newborn baby about that. The second difficulty is to be ensure there are not any deadlocks. I ensured that by organizing send and the receives by the opposite order. The third difficulty was to set up environment since there is not enough material in the internet about mpi4py.

Conclusion

The program is implemented as desired but there can be some improvements. Using MPI interface has some difficulties but the multiprocessing improves the time of the program. However if the worker count is too much, there would be too many interactions between the workers which is bad point. Thus 5, 17, 37 are the best number of processors for this program.

Appendices

Code

```
from mpi4py import MPI
import numpy as np
import math
import sys

#Initilazing mpi and getting rank
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
#Getting necessary variables
Grid_Size = 360
worker_count = comm.Get_size()-1
w_per_dimension = int(math.sqrt(worker_count))
Rank_size = Grid_Size//w_per_dimension
time = int(sys.argv[3])
#method to get rank from col and row
def to_rank(row, col):
    row = row % w_per_dimension
    col = col % w_per_dimension
    return row * w_per_dimension + col + 1
#method to get row and col from rank
def from_rank(rank):
    row = (rank - 1) // w_per_dimension
    col = (rank - 1) % w_per_dimension
    return (row, col)
#calculating the next value of a square according to game of life rules
def next_sqr(center, sum):
    if center == 0:
        if sum == 3:
            return 1
        else:
            return 0
    else:
```

```python
        if sum < 2 or sum > 3:
            return 0
        else:
            return 1
#a method to calculate the next situation of grid of a worker
def next_step(upleft, up, upright, downleft, down, downright, left, right, dat
a):
    #New values will be here
    new_data = np.zeros((Rank_size,Rank_size), dtype=int)
    #Concatenating the matrices
    top = np.concatenate((upleft, up, upright), axis=1)
    middle = np.concatenate((left, data, right), axis=1)
    bot = np.concatenate((downleft, down, downright), axis=1)
    #The extended matrix
    yeni = np.concatenate((top, middle, bot), axis=0)
    #Calculating all the next values of the matrix
    for i in range(Rank_size):
        new_data[i] = [next_sqr(data[i][j], yeni[i][j]+ yeni[i][j+1]+ yeni[i][
j+2]+ yeni[i+1][j]+ yeni[i+1][j+2]+ yeni[i+2][j]+ yeni[i+2][j+1]+ yeni[i+2][j+
2]) for j in range(Rank_size)]
    return new_data

#Beginning of the process
#Master organizing the input
if rank == 0:
    #Getting the arguments
    input = sys.argv[1]
    output = sys.argv[2]
    #Loading the input
    grid = np.loadtxt(input, dtype=int)
    #Partition of the input and sending it to workers
    for i in range(1, worker_count + 1):
            row = from_rank(i)[0]
            col = from_rank(i)[1]
            data = np.copy(grid[row*Rank_size : (row+1)*Rank_size, col*Rank_si
ze : (col+1)*Rank_size])
            comm.Send(data, dest = i)
#If not master than calculate the row and col, receive the matrix
else:
    row = from_rank(rank)[0]
    col = from_rank(rank)[1]
    data = np.zeros((Rank_size, Rank_size), dtype=int)
    comm.Recv(data, source = 0)
#Let the process begins
for t in range(time):
    #There is nothing to do for master here
    if rank != 0:
        #Creating empty arrays to be received from other workers
        right = np.zeros((Rank_size, 1), dtype=int)
        left = np.zeros((Rank_size, 1), dtype=int)
        down = np.zeros((1, Rank_size), dtype=int)
        up = np.zeros((1, Rank_size), dtype=int)
        upright = np.zeros((1,1), dtype=int)
        upleft = np.zeros((1,1), dtype=int)
        downright = np.zeros((1,1), dtype=int)
        downleft = np.zeros((1,1), dtype=int)
        #If it is a white square in Chess, it sends first
        if (row + col) % 2 == 0:
            #Sending left, right, up, down respectively
            comm.Send(np.copy(data[:, :1]), dest = to_rank(row, col - 1))
            comm.Send(np.copy(data[:, -1:]), dest = to_rank(row, col + 1))
            comm.Send(np.copy(data[:1, :]), dest = to_rank(row - 1, col))
            comm.Send(np.copy(data[-1:, :]), dest = to_rank(row + 1, col))
            #Receiving right, left, down, up respectively
            #Which are the opposite of the sending order in order to prevent D
eadlocks
            comm.Recv(right, source = to_rank(row, col + 1))
            comm.Recv(left, source = to_rank(row, col - 1))
            comm.Recv(down, source = to_rank(row + 1, col))
            comm.Recv(up, source = to_rank(row - 1, col))
        #If it is a black square
        else:
            #Receiving right, left, down, up respectively
            #Which are the opposite of the sending order in order to prevent D
eadlocks
            comm.Recv(right, source = to_rank(row, col + 1))
            comm.Recv(left, source = to_rank(row, col - 1))
            comm.Recv(down, source = to_rank(row + 1, col))
            comm.Recv(up, source = to_rank(row - 1, col))
            #Sending left, right, up, down respectively
            comm.Send(np.copy(data[:, :1]), dest = to_rank(row, col - 1))
            comm.Send(np.copy(data[:, -1:]), dest = to_rank(row, col + 1))
            comm.Send(np.copy(data[:1, :]), dest = to_rank(row - 1, col))
            comm.Send(np.copy(data[-1:, :]), dest = to_rank(row + 1, col))
        #It is muffin time!
        #Sending the corners according to col is odd or not
        if col % 2 == 0:
            #Sending upleft, upright, downleft, downright respectively
            comm.Send(np.copy(data[:1, :1]), dest = to_rank(row - 1, col - 1))
            comm.Send(np.copy(data[:1, -
1:]), dest = to_rank(row - 1, col + 1))
            comm.Send(np.copy(data[-
1:, :1]), dest = to_rank(row + 1, col - 1))
            comm.Send(np.copy(data[-1:, -
1:]), dest = to_rank(row + 1, col + 1))
            #Receiving upleft, upright, downleft, downright respectively
            #Which are the opposite of the sending order in order to prevent D
eadlocks
```

```
            comm.Recv(downright, source = to_rank(row + 1, col + 1))
            comm.Recv(downleft, source = to_rank(row + 1, col - 1))
            comm.Recv(upright, source = to_rank(row - 1, col + 1))
            comm.Recv(upleft, source = to_rank(row - 1, col - 1))
        else:
            #Receiving upleft, upright, downleft, downright respectively
            #Which are the opposite of the sending order in order to prevent D
eadlocks again
            comm.Recv(downright, source = to_rank(row + 1, col + 1))
            comm.Recv(downleft, source = to_rank(row + 1, col - 1))
            comm.Recv(upright, source = to_rank(row - 1, col + 1))
            comm.Recv(upleft, source = to_rank(row - 1, col - 1))
            #Sending upleft, upright, downleft, downright respectively
            comm.Send(np.copy(data[:1, :1]), dest = to_rank(row - 1, col - 1))
            comm.Send(np.copy(data[:1, -
1:]), dest = to_rank(row - 1, col + 1))
            comm.Send(np.copy(data[-
1:, :1]), dest = to_rank(row + 1, col - 1))
            comm.Send(np.copy(data[-1:, -
1:]), dest = to_rank(row + 1, col + 1))
        #Join all the info gathered from other workers and calculate the new m
atrix
        data = next_step(upleft, up, upright, downleft, down, downright, left,
 right, data)
#Lets finish the process
#Master collecting the data and saving it
if rank == 0:
    new_grid = np.zeros((Grid_Size,Grid_Size), dtype=int)
    #getting data
    for i in range(1, worker_count + 1):
        row = from_rank(i)[0]
        col = from_rank(i)[1]
        data = np.zeros((Rank_size,Rank_size), dtype=int)
        comm.Recv(data, source = i)
        new_grid[row*Rank_size : (row+1)*Rank_size, col*Rank_size : (col+1)*Ra
nk_size] = data
    #saving the data
    np.savetxt(output, new_grid, delimiter=' ', fmt='%1d', newline=' \n')
#Workers sending their data to master
else:
    comm.Send(np.copy(data), dest = 0)
```