Programming Project
MPI Parallel Game of Life

Modular - Looped - Checkered - Periodic

CmpE 300 - ANALYSIS OF ALGORITHMS

Taha Eyup Korkmaz – 2014400258

BOGAZICI UNIVERSITY 21.12.2019

Taha Eyup Korkmaz
2014400258
20.12.2019

CmpE 300 - ANALYSIS OF ALGORITHMS

Programming Project - MPI Parallel Game of Life

Modular - Looped - Checkered - Periodic

Taha Eyup Korkmaz – 2014400258

– Introduction

Our focus will be on the particular cellular automaton called the (Conway's) Game of Life, devised by J. H. Conway in 1970. Just "Game" for short. In the Game, we have a 2dimensional orthogonal grid as a map (i.e. a matrix). Each cell on the map can either contain a creature (1) or be empty (0). As stated I implemented an advanced version of the game of life which contains the initial rules.

The rules were:

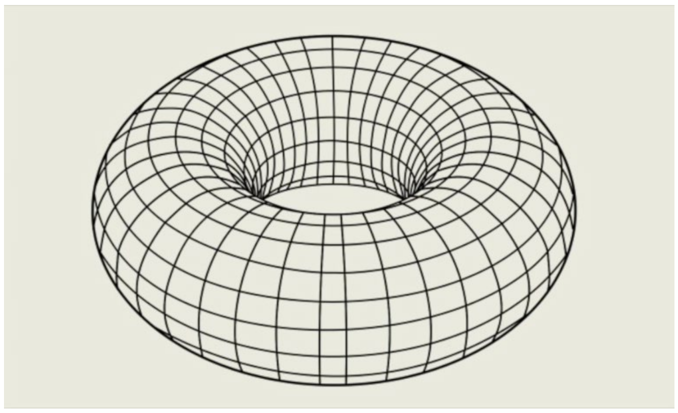
- ***Loneliness kills:*** A creature dies (i.e. the cell becomes empty) if it has less than 2 neighboring creatures.
- ***Overpopulation also kills:*** A creature dies (and becomes empty) if it has more than 3 neighboring creatures. See the following example. Note that the creature would not die if it had one less neighboring creature.
- ***Reproduction:*** A new life appears on an empty cell if it has exactly 3 neighboring creatures. See the following example. Note that the creature would not be born if the cell had one more or one less neighboring creature
- ***Otherwise:*** In any other condition, the creatures remain alive, and the empty spaces remain empty

The added new spesifications are:

- ***Checkered:*** The map will be split into C square arrays, each with S/√C rows and columns. The following image shows how this split would be with S = 36 and C = 16.

- ***Periodic:*** The neighbors are not missing, when they're on the sides. It's just the other corner so the whole 2D array is actually connected as it's called toroidal shapes which the right end is connected to left and top is connected to bottom as we've implemented.

Taha Eyup Korkmaz
2014400258 20.12.2019



As for the Project, I implemented the Game of Life in a MPI solution that handled by C+1 amount of mini processors that work independantly and communicate each other when necessary. The input of was split into parts by the manager processor which is the rank=0 and sent to the slaves and they were calculating the game itself. After certain number of Iterations the slaves send the result to manager and manager combines the end results and writes into the output.

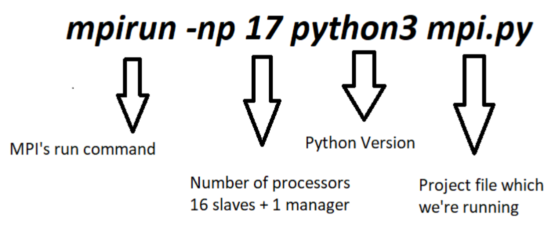
- **Program Interface**

The program is a Python program which should be run in a python environment. I used the Python 3 some of the usages might not be avaliable in the Python 2 so its recomended to use the run it on Python 3. It also uses numpy and mpi4py libraries so they should be installed as well. To execute the program, user must simply open the terminal in the directory and there should be a txt file that has the nxn matrix with 1s and 0s using the “ “ blank space as the delimeter.

The program will run the Game of Life Iterations number of times. User also has to specify in the command how much processors should be working. So an example running command should be like:

```
mpirun -np 17 python3 mpi.py
```

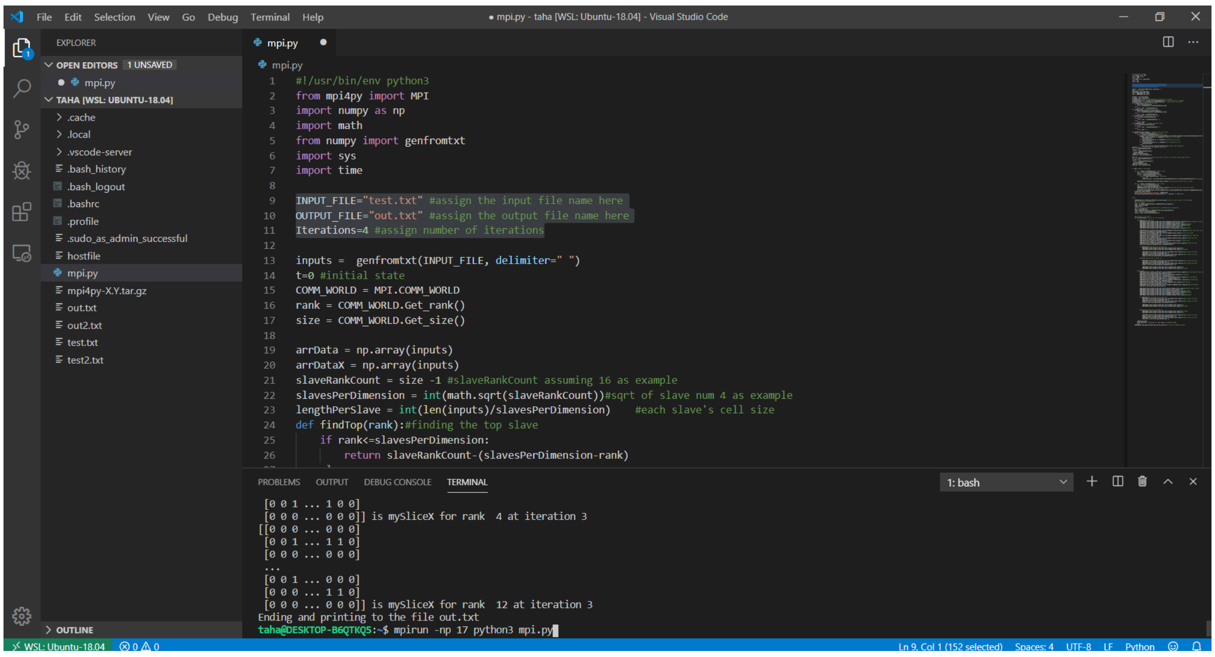
Taha Eyup Korkmaz
2014400258 20.12.2019



With 17 is the number of processors used as slaves which is 16 here + 1, the manager. Also the "test.txt" file's row and column number should be able get divided by the square root of the used slave number that 4 here.

– Program Execution

The input file name, output file name and the iteration name should be supplied by the user at the start of the code which you can see below. The input and the output file names are set as "test.txt" and "out.txt" respectively at the top of the code as INPUT_FILE and OUTPUT_FILE. They can be changed accordingly. Also user can set the Iteration of the game with changing the Iterations variable.

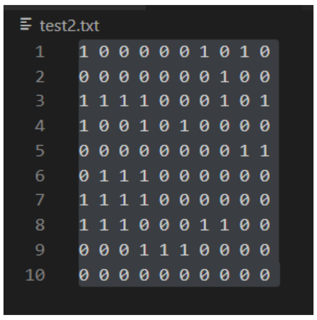


```
#!/usr/bin/env python3
from mpi4py import MPI
import numpy as np
import math
from numpy import genfromtxt
import sys
import time

INPUT_FILE="test.txt" #assign the input file name here
OUTPUT_FILE="out.txt" #assign the output file name here
Iterations=4 #assign number of iterations

inputs =  genfromtxt(INPUT_FILE, delimiter=" ")
t=0 #initial state
COMM_WORLD = MPI.COMM_WORLD
rank = COMM_WORLD.Get_rank()
size = COMM_WORLD.Get_size()

arrData = np.array(inputs)
arrDataX = np.array(inputs)
slaveRankCount = size -1 #slaveRankCount assuming 16 as example
slavesPerDimension = int(math.sqrt(slaveRankCount))#sqrt of slave num 4 as example
lengthPerSlave = int(len(inputs)/slavesPerDimension)    #each slave's cell size
def findTop(rank):#finding the top slave
    if rank<=slavesPerDimension:
        return slaveRankCount-(slavesPerDimension-rank)
```

```
 [0 0 1 ... 1 0 0]
 [0 0 0 ... 0 0 0]] is mySliceX for rank  4 at iteration 3
[[0 0 0 ... 0 0 0]
 [0 0 1 ... 1 1 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 1 ... 0 0 0]
 [0 0 0 ... 1 1 0]
 [0 0 0 ... 0 0 0]] is mySliceX for rank  12 at iteration 3
Ending and printing to the file out.txt
taha@DESKTOP-B6QTKQ5:~$ mpirun -np 17 python3 mpi.py
```

– Input and Output

Input file should be txt file and should be in nxn matrix format which is like this:


test2.txt

```
1 0 0 0 0 0 1 0 1 0
0 0 0 0 0 0 0 1 0 0
1 1 1 1 0 0 0 1 0 1
1 0 0 1 0 1 0 0 0 0
0 0 0 0 0 0 0 0 1 1
0 1 1 1 0 0 0 0 0 0
1 1 1 1 0 0 0 0 0 0
1 1 1 0 0 0 1 1 0 0
0 0 0 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

It's a 10x10 matrix

The output file will be at the same format as the input file, 10x10 in this case. It will look exactly like the input.

– Program Structure

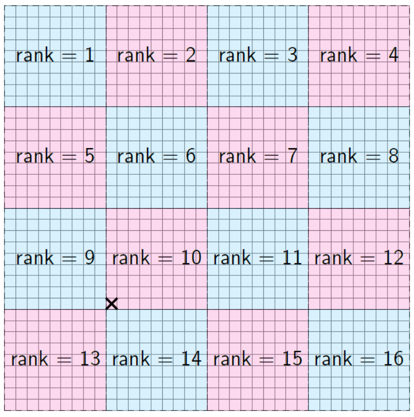
Subprograms:

- Game of Life (mySlice)

This function requests a mySlice parameter from the slave that has the slave's square and the neighbors added to the top/bot rows and right/left colunms. The function calculates the new state of the game with the rules stated above and stores them in a new array called mySliceX that we'll be using later to get the new state.

- Find Top/Bot/Right/Left (Rank)

The names of the function actually tells the story. This is where the the Project gets toroidal because we'll be setting the 1s row's upper row as 13 if we think as 16 slaves are working see the image below for representation.



The If/Else that handles the manager and slaves

- If rank = 0:

It’s the if that we’re doing the manager’s jobs here, which are divide the input and distribute it into the slaves, after the slaves do their jobs then receive the last state of the game and print it into the output file.

- else:

It’s the place where all the slaves are doing their jobs and communicate with each other. They receive the initial input from the manager and do the calculations iteration times and send it back to the manager after they’re execute the Game of Life.

<u>All arrays that used in the program</u>

- sendSlice

The array that distributed among the processes

- mySliceX

The array that the new state gets stored temporarily

- temp

The array that used to calculate the game of life. It has a n+2xn+2 matrix that includes the neighboring cells’s info too.

- arrData

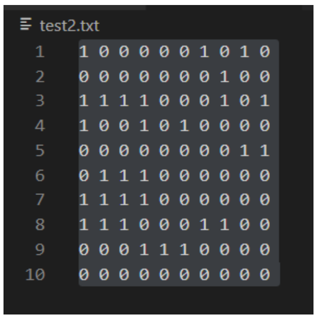
Initial state of the game which is basically the input file stored in an array

- arrDataX

The array that distributed among the processes

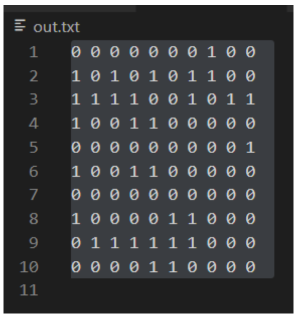
– **Examples**

As an example, lest calculate a small inputs Game of Life for 2 Iterations.


test2.txt

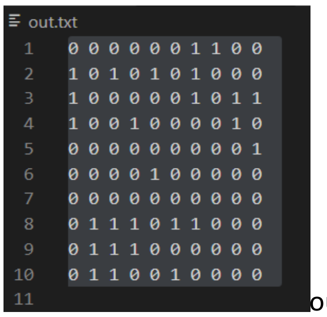
```
1 0 0 0 0 0 1 0 1 0
0 0 0 0 0 0 0 1 0 0
1 1 1 1 0 0 0 1 0 1
1 0 0 1 0 1 0 0 0 0
0 0 0 0 0 0 0 0 1 1
0 1 1 1 0 0 0 0 0 0
1 1 1 1 0 0 0 0 0 0
1 1 1 0 0 0 1 1 0 0
0 0 0 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

test


out.txt

```
0 0 0 0 0 0 0 1 0 0
1 0 1 0 1 0 1 1 0 0
1 1 1 1 0 0 1 0 1 1
1 0 0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1
1 0 0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 1 1 0 0 0
0 1 1 1 1 1 1 0 0 0
0 0 0 0 1 1 0 0 0 0
```

output


out.txt

```
0 0 0 0 0 0 1 1 0 0
1 0 1 0 1 0 1 0 0 0
1 0 0 0 0 0 1 0 1 1
1 0 0 1 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 1 1 1 0 1 1 0 0 0
0 1 1 1 0 0 0 0 0 0
0 1 1 0 0 1 0 0 0 0
```

output2

We can see the top right 9th 1 at the test.txt that died in the next state but the 8th one is came to life. Lets see the next iteration. We can consider the first output is the second one's input. We can see the 7th one at the first row came to life this time as well. Because it as 3 1s in the neighborhood, so it's now a 1 too.

– **Improvements and Extensions**

First deviation for me was how to handle the communication because the send and receive are kind of hardwired, so as a developer I had to state exactly where to send and where to receive in very strict manner and use it efficiently as possible.

As an improvement we might add a feature that shows the whole state of game as it happens but we were restricted about the communication of the slaves and the manager so that couldn't be implemented here.

Also we can add another feature that user can manually add a 1 to another cell while the game is happening after we implement the first feature. It would help the users to understand how game of life kills or creates new cells as they'll be interacting with the game while it's happening

– **Difficulties Encountered**

Obviously the first diffuculty was how the slaves will communicate without getting a deadlock. After that where to store the received information and how to use it. The multiprocessor logic is somehow hard to debug and because of that it was hard to identify the problems. I had to completely start over the Project after realizing that my communication plan wasn't working and I had to modularize the Project because it was so hard to read the code.

Other problem was getting used to Python syntax and method's differences with the C++ and Java that I accustomed to. For instance I didn't know array = array is soft copy in Python, so I had to use the numpy's copy method to hard copy the values inside.

Finally the calendar management. I am actually very eager to talk about this in the academic committee meeting of our department. I was between suicide, giving up about the course, trying to enjoy the life, do the other course's Project, study for another course's mid term or it's 4th Project. It seemed that our department tries to make us commit suicide or send us into existential crisises and wants to make us feel extremely stupid. Between a lot of very talented group of people, <u>it's very bad that our department makes us get bad grades and also make us insecure at the same time so that we can fail at the life after graduation if we manage to graduate.</u> It's okey to that we need to study a lot, the problem is we can't go to Erasmus/Exchange or do a master's at a good place after a trainwreck of grades. It just feels so bad seeing many Management studying friend that just party in their life that have above 3 gpa while many of us are struggling to climb up even the 2.5 number. I'm not even talking about many other departments. I'm pretty sure our departmen is at bottom 3 at our university about average department GPA. I think head of department can check whether if its true or not. Regardless it's so bad that many of my collagues are considering/attempted suicide due to academic pressure and intensity.

To change something, do some us really has to sacrifice himself/herself?

– Conclusion

After many hours, I realized the multiprocessor design is a very complex and hard topic. Not only it's very hard to debug, it's also very challanging to make sure the information is distributed correctly and used properly.

I think I've learned that using modules and writing clean code is very helpfull since we'll be dealing with a lot of complex problems and it's really hard to read and actually write otherwise.

Also not writing hardwired code is necessary as well. I saw a lot of my friends were writing as indices regarding about 360 number specified in the description but it's not a good design choice since we might have to change the inputs for testing or for something else. Using some global variables and functions also helps trumendously.

– Appendices

Taha Eyup Korkmaz
2014400258 20.12.2019

You can see the source code below. There are comments too make the reading clear.

```
#!/usr/bin/env python3
from mpi4py import MPI
import numpy as np
import math
from numpy import genfromtxt
import sys
import time

INPUT_FILE="test.txt" #assign the input file name here
OUTPUT_FILE="out.txt" #assign the output file name here
Iterations=10 #assign number of iterations

arrData =  genfromtxt(INPUT_FILE, delimiter=" ")
t=0 #initial state
COMM_WORLD = MPI.COMM_WORLD
rank = COMM_WORLD.Get_rank()
size = COMM_WORLD.Get_size()

arrData = np.array(arrData)
arrDataX = np.array(arrData)
slaveRankCount = size -1 #slaveRankCount assuming 16 as example
slavesPerDimension = int(math.sqrt(slaveRankCount))#sqrt of slave num 4 as example
lengthPerSlave = int(len(arrData)/slavesPerDimension)    #each slave's cell size
def findTop(rank):#finding the top slave
    if rank<=slavesPerDimension:
        return slaveRankCount-(slavesPerDimension-rank)
    else:
        return rank - slavesPerDimension
def findBot(rank):  #finding the bot slave
    if rank>slaveRankCount-slavesPerDimension:
        return slavesPerDimension-(slaveRankCount-rank)
    else:
        return rank + slavesPerDimension
def findLeft(rank): #finding the left slave
    a = int(rank-1)%slavesPerDimension
    if a == 0:
        return rank + slavesPerDimension - 1
    else:
        return rank-1
def findRight(rank):    #finding the Right slave
    a = int(rank)%slavesPerDimension
    if a == 0:
        return rank - slavesPerDimension + 1
    else:
        return rank + 1

def gameOfLife(mySliceInput):   #game of life calculation
```

```
    for j in range(1,lengthPerSlave+1):#for rows
        for k in range(1,lengthPerSlave+1):#for columns
            neighbor = mySliceInput[j-1][k-1]+mySliceInput[j-1][k]+mySliceInput[j-
1][k+1]+mySliceInput[j][k-1]+mySliceInput[j][k+1]+mySliceInput[j+1][k-
1]+mySliceInput[j+1][k]+mySliceInput[j+1][k+1]#calculating the sum of 1s in neighbors
            if mySliceInput[j][k]==0 and neighbor==3: #balance in all things
                mySliceX[j][k]=1
            elif mySliceInput[j][k]==1 and neighbor<2:#loneliness also kills
                mySliceX[j][k]=0
            elif  mySliceInput[j][k]==1 and neighbor>3: # overpopulation kills
                mySliceX[j][k]=0
            else:
                mySliceX[j][k]=np.copy(mySliceInput[j][k]) #remain same otherwise
    temp[1:lengthPerSlave+1,1:lengthPerSlave+1]=np.copy(mySliceX[1:lengthPerSlave+1,1:lengthPerSlave+1]
)
    sendSlice=np.copy(mySliceX[1:lengthPerSlave+1,1:lengthPerSlave+1])


sendSlice = [] #initialization the initial sent slice
for i in range(lengthPerSlave):
 column = []
 for j in range(lengthPerSlave):
  column.append(0)
 sendSlice.append(column)
sendSlice = np.array(sendSlice)

mySliceX = [] #initialization the big array that includes the recieved top/up/right/left etc
for i in range(lengthPerSlave+2):
 column = []
 for j in range(lengthPerSlave+2):
  column.append(0)
 mySliceX.append(column)
mySliceX = np.array(mySliceX)


if rank == 0:#for the manager

    for i in range(1,slaveRankCount+1): ##for slaves
        row = int((i-1)/slavesPerDimension)
        column = i-(row*slavesPerDimension)-1
        for j in range(lengthPerSlave):  ##rows
            for k in range(lengthPerSlave):  ##columns
                #print(k)
                sendSlice[j][k] = np.copy(arrData[(row*lengthPerSlave)+j][(column*lengthPerSlave)+k]) #
dividing the input to every slave

        COMM_WORLD.Send([sendSlice,MPI.INT],dest=i,tag=14) #sending the divided arrays to slaves

    for i in range(1,slaveRankCount+1): #for slaves
```

```
        row = int((i-1)/slavesPerDimension)
        column = i-(row*slavesPerDimension)-1
        COMM_WORLD.Recv([mySliceX,MPI.INT],source=i,tag=14) #receiving the last states from slaves
        #print(mySliceX,'came to the manager from ',i)
        arrDataX[row*lengthPerSlave:(row+1)*lengthPerSlave,column*lengthPerSlave:(column+1)*lengthPerSl
ave]=np.copy(mySliceX[1:lengthPerSlave+1,1:lengthPerSlave+1]) #setting the end state
    arrData=np.copy(arrDataX)
    print('Ending and printing to the file', OUTPUT_FILE)
    np.savetxt(fname=OUTPUT_FILE,X=arrData,delimiter=" ", newline=" \n",fmt='%1d')


else:

    COMM_WORLD.Recv([sendSlice,MPI.INT],source=0,tag=14) #initial state recevie from the manager
    temp = np.empty((lengthPerSlave+2,lengthPerSlave+2),dtype=int)
    temp = np.array(temp)
    temp.fill(0)
    len = len(sendSlice)
    temp[1:lengthPerSlave+1,1:lengthPerSlave+1]=np.copy(sendSlice)
    row = (rank-1)//slavesPerDimension
    column = rank-(row*slavesPerDimension)-1

    while(t<Iterations): # while for the iterations

        if rank%2==1:#For Odd Ones
            COMM_WORLD.Send([sendSlice,MPI.INT],dest=findRight(findTop(rank)),tag=14)#RightTop sending
to evens
            COMM_WORLD.Send([sendSlice,MPI.INT],dest=findRight(rank),tag=14)#Right
            COMM_WORLD.Send([sendSlice,MPI.INT],dest=findRight(findBot(rank)),tag=14)#RightBot
            COMM_WORLD.Send([sendSlice,MPI.INT],dest=findLeft(findBot(rank)),tag=14)#LeftBot
            COMM_WORLD.Send([sendSlice,MPI.INT],dest=findLeft(rank),tag=14)#Left
            COMM_WORLD.Send([sendSlice,MPI.INT],dest=findLeft(findTop(rank)),tag=14)#LeftTop

            COMM_WORLD.Recv([sendSlice,MPI.INT],source=findRight(findTop(rank)),tag=14)#receive from ri
ght top even
            temp[0,len+1]=sendSlice[0,lengthPerSlave-1]
            COMM_WORLD.Recv([sendSlice,MPI.INT],source=findRight(rank),tag=14)#receive from right
            temp[1:len+1,len+1]=sendSlice[:,0]
            COMM_WORLD.Recv([sendSlice,MPI.INT],source=findBot(findRight(rank)),tag=14)#receive right b
ot
            temp[len+1,len+1]=sendSlice[0,0]
            COMM_WORLD.Recv([sendSlice,MPI.INT],source=findBot(findLeft(rank)),tag=14)#receive bot left
            temp[0,len+1]=sendSlice[0,lengthPerSlave-1]
            COMM_WORLD.Recv([sendSlice,MPI.INT],source=findLeft(rank),tag=14)# receive left
            temp[0,1:len+1]=sendSlice[:,lengthPerSlave-1]
            COMM_WORLD.Recv([sendSlice,MPI.INT],source=findTop(findLeft(rank)),tag=14)#receive top left
            temp[0,0]=sendSlice[lengthPerSlave-1,lengthPerSlave-1]

            if row%2==0:#Odd and Row even ie first row
```

```
                COMM_WORLD.Send([sendSlice,MPI.INT],dest=findTop(rank),tag=14)#Top
                COMM_WORLD.Send([sendSlice,MPI.INT],dest=findBot(rank),tag=14)#Bot Sends

                COMM_WORLD.Recv([sendSlice,MPI.INT],source=findBot(rank),tag=14)#tag14 receive from Bot
                temp[len+1,1:len+1]=np.copy(sendSlice[0,:])
                COMM_WORLD.Recv([sendSlice,MPI.INT],source=findTop(rank),tag=14)#tag14 receive from Top
                temp[0,1:len+1]=np.copy(sendSlice[lengthPerSlave-1,:])

            else:#Odd and Row odd ie second row

                COMM_WORLD.Recv([sendSlice,MPI.INT],source=findBot(rank),tag=14)#tag14 receive from Bot
                temp[len+1,1:len+1]=np.copy(sendSlice[0,:])
                COMM_WORLD.Recv([sendSlice,MPI.INT],source=findTop(rank),tag=14)#tag14 receive from Top
                temp[0,1:len+1]=np.copy(sendSlice[lengthPerSlave-1,:])

                COMM_WORLD.Send([sendSlice,MPI.INT],dest=findTop(rank),tag=14)#Top
                COMM_WORLD.Send([sendSlice,MPI.INT],dest=findBot(rank),tag=14)#Bot Sends

        else: #even
            COMM_WORLD.Recv([sendSlice,MPI.INT],source=findBot(findLeft(rank)),tag=14)#Left Bot Receive
 from the odds
            temp[len+1,0]=np.copy(sendSlice[0,lengthPerSlave-1])
            COMM_WORLD.Recv([sendSlice,MPI.INT],source=findLeft(rank),tag=14)#Left Receive
            temp[1:len+1,0]=np.copy(sendSlice[:,lengthPerSlave-1])
            COMM_WORLD.Recv([sendSlice,MPI.INT],source=findTop(findLeft(rank)),tag=14)#Left Top Receive
            temp[0,0]=np.copy(sendSlice[lengthPerSlave-1,lengthPerSlave-1])
            COMM_WORLD.Recv([sendSlice,MPI.INT],source=findRight(findTop(rank)),tag=14)#Rigt Top Receiv
e
            temp[0,len+1]=np.copy(sendSlice[0,lengthPerSlave-1])
            COMM_WORLD.Recv([sendSlice,MPI.INT],source=findRight(rank),tag=14)#Right Receive
            temp[1:len+1,len+1]=np.copy(sendSlice[:,0])
            COMM_WORLD.Recv([sendSlice,MPI.INT],source=findBot(findRight(rank)),tag=14)#Right Bot Recei
ve
            temp[len+1,len+1]=np.copy(sendSlice[0,0])

            COMM_WORLD.Send([sendSlice,MPI.INT],dest=findLeft(findBot(rank)),tag=14)#LeftBot sends to O
dds
            COMM_WORLD.Send([sendSlice,MPI.INT],dest=findLeft(rank),tag=14)#Left
            COMM_WORLD.Send([sendSlice,MPI.INT],dest=findLeft(findTop(rank)),tag=14)#LeftTop
            COMM_WORLD.Send([sendSlice,MPI.INT],dest=findRight(findTop(rank)),tag=14)#RightTop
            COMM_WORLD.Send([sendSlice,MPI.INT],dest=findRight(rank),tag=14)#Right
            COMM_WORLD.Send([sendSlice,MPI.INT],dest=findRight(findBot(rank)),tag=14)#RightBot

            if row%2==0:#Even and even row ie first row
                COMM_WORLD.Recv([sendSlice,MPI.INT],source=findBot(rank),tag=14)#tag14 receive from Bot
                temp[len+1,1:len+1]=np.copy(sendSlice[0,:])
                COMM_WORLD.Recv([sendSlice,MPI.INT],source=findTop(rank),tag=14)#tag14 receive from Top
                temp[0,1:len+1]=np.copy(sendSlice[lengthPerSlave-1,:])
```

Taha Eyup Korkmaz
2014400258 20.12.2019

```
                COMM_WORLD.Send([sendSlice,MPI.INT],dest=findBot(rank),tag=14)#Top
                COMM_WORLD.Send([sendSlice,MPI.INT],dest=findTop(rank),tag=14)#Bot Sends



            else:#even and odd row ie second row
                COMM_WORLD.Send([sendSlice,MPI.INT],dest=findTop(rank),tag=14)#Bot Sends

                COMM_WORLD.Send([sendSlice,MPI.INT],dest=findBot(rank),tag=14)#Top

                COMM_WORLD.Recv([sendSlice,MPI.INT],source=findTop(rank),tag=14)#tag14 receive from Top
                temp[0,1:len+1]=np.copy(sendSlice[lengthPerSlave-1,:])
                COMM_WORLD.Recv([sendSlice,MPI.INT],source=findBot(rank),tag=14)#tag14 receive from Bot
                temp[len+1,1:len+1]=np.copy(sendSlice[0,:])

        gameOfLife(temp)
        print(mySliceX,'is mySliceX for rank',rank,'at iteration',t+1)
        t+=1
    COMM_WORLD.Send([mySliceX,MPI.INT],dest=0,tag=14)#Return Back to Manager(rank=0)
```