HW08 - Game of Life

CS5500

Chelsi Gupta

October 27, 2018

1 Steps for Implementation

- I have made 5 functions in my program:
  - **generate_organism:**
    Output- 0 or 1
    Purpose- It uses the pseudo random number generator i.e. random.random() to seed a cell with organism so that there is a 1-in-5 probability that it contains an organism.
  - **init_world:**
    Output- world(1024*1024 numpy array)
    Purpose- Creates a 1024*1024 numpy array where the probability of a cell containing an organism is 0.2.
  - **give_chunk:**
    Input- data(world), chunk_no(rank), size
    Output- data(chunk for process)
    Purpose- Provides appropriate chunk to each process. Where all processes(except 0 and size-1) get equal sized chunks.Each process(except 0 and size-1) gets two extra rows one from rank-1 and other from rank + 1.
  - **survival:**
    Input- x, y(coordinates of cell), world
    Output- world[value of cell]
    Purpose- Checks the neighbors of a cell and decides if the cell lives to next generation by applying the rules of game of life.
  - **generation:**
    Input- world,rank,size
    Output- new_world
    Purpose- Applies the survival function to each cell in the world.
- The rank 0 first creates an initial world by calling init_world, which it append to a list called ims.

- Process 0 then sends appropriate chunk to each process, which in turn calculates the generation for its chunk and returns it back to process 0.
- Process 0 then merges the chunks from all processes and after 100 iterations, saves the list of images as a gif file.
- I compiled and ran the program using 'mpirun –oversubscribe -np 9 python parallel_game_of_life.py'. This saved the output to Parallel_gof.gif file.

2 Code

```
import time
start_time = time.time()
import numpy as np
import matplotlib.pyplot as plt
import random
import matplotlib.animation as animation
from mpi4py import MPI
comm = MPI.COMM_WORLD     # Defines the default communicator


def generate_organism():
        org = random.random()
        if org > 0.2:
                return 0
        return 1


def init_world():
        world = np.zeros((1024, 1024))
        for i in range(0,len(world),1):
                for j in range(0,len(world[i]),1):
                        world[i,j] = generate_organism()
        return world


def give_chunk(data, chunk_no, size):
        lower_index= (chunk_no-1)*(len(data)//size)
        upper_index= chunk_no*(len(data)//size) -1
        if(chunk_no>1):
                lower_index-=1
        if(chunk_no<size):
                upper_index+=1

        return data[lower_index:upper_index+1]


def survival(x, y, world):
        num_neighbours = np.sum(world[x - 1 : x + 2, y - 1 : y + 2]) - world[x, y]
        # The rules of Life
        if (world[x, y] == 1) and (num_neighbours not in (2,3)):
                return 0
        elif num_neighbours == 3:
                return 1
```

```
        return world[x, y]


def generation(world,rank,size):
        new_world = np.copy(world)
        # Apply the survival function to every cell in the universe
        for i in range(0,len(world),1):
                for j in range(0,len(world[i]),1):
                        new_world[i, j] = survival(i, j, world)

        lower_index= 0
        upper_index= len(world)-1
        if(rank>1):
                lower_index=1
        if(rank<size-1):
                upper_index-=1

        return new_world[lower_index:upper_index+1]


size = comm.Get_size()  # Stores the number of processes in size.
rank = comm.Get_rank()  # Stores the rank (pid) of the current process

if rank == 0:
        fig = plt.figure()
        ims = []
        world=init_world()
        im =plt.imshow(world, cmap='binary')
        ims.append([im])
        for iii in range(100):
                for i in range(1,size):
                        chunk=give_chunk(world,i,size-1)
                        comm.send(chunk,dest=i)

                world=[]
                for i in range(1,size):
                        chunk=comm.recv(source=i)
                        world.append(chunk)

                world = np.vstack(world)
                im =plt.imshow(world, cmap='binary')
                ims.append([im])
        ani = animation.ArtistAnimation(fig, ims, interval=100, blit=True, repeat_delay=10)
        ani.save("Parallel_gof.gif", writer="imagemagick")
        print("--- %s seconds ---" % (time.time() - start_time))

else:
        for iii in range(100):
                chunk=comm.recv(source=0)
                chunk=generation(chunk,rank,size)
                comm.send(chunk,dest=0)
```

3 Output

Output is a .gif file and so it cannot be included here. I have mailed it to the TA and professor separately. The output is in Parallel_gof.gif file.

4 Timing Information

I tried running the program with different number of processors and the result is as follows:

#Processors used	**Time-taken(in seconds)**
9	368.15
17	360.03
33	368.00
65	393.16
129	483.96

References

- https://medium.com/@martin.robertandrew/conways-game-of-life-in-python-2900a6dcdc97
- https://github.com/robertmartin8/PyGameofLife/blob/master/life.py