CMPE 300 Analysis of Algorithms

MPI Project Documentation

Alperen Bağ
2018400279

Submission Date
17.12.2019

Introduction

In this project Conway's Game of Life is simulated with parallel programming. For the parallel programing environment, Open MPI library is used. The rules of the game can be found here.

Actually, the implementation of the game is pretty simple, because the rules are not very complicated. However, a big-scale input for the initial state of the game can be very tiring for the CPU. Hence, we should benefit the parallelizable nature of the game and utilize parallel programming paradigm, which is the case in this project. The project consists of only one code file which is *main.cpp.* How to compile and run the code will be explained in the following section.

Program Interface

To compile *main.cpp:*

mpic++ main.cpp -o game

To run:

mpirun -np [M] --oversubscribe ./game input.txt output.txt [T]

Here, "input.txt" is the name of the input file, "output.txt" is the name of the output file. While [M] represents the number of cores we want to use to run the game, [T] is the number of iterations we want to run.

Program Execution

In order to run the program, user should decide the number of processors and the number of iterations. Users should be aware of that the number of worker processors is one less than the parameter [M], because one processor is used as manager. When user prepare the input file, they can run the program with the above

mentioned commands. Format of the input file will be explained in detail in the following section.

When the program terminated successfully, output file will contain the new map after [T] iterations.

Input and Output

Input format is as following,

- Every cell on a row is separated with a space character.
- Every row is separated with new line character (\n). (Last row contains \n, as well.)
- 0 represents that the cell is dead, 1 represents that the cell is alive.

Output file will contain the final map after [T] iterations and its format will be the same as the input file.

Program Structure

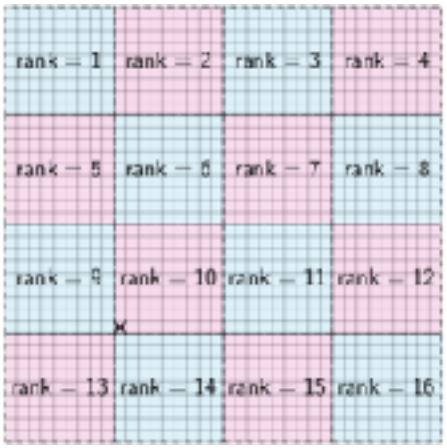
1- FUNCTIONS

create_map: Initializes the map by reading the input file

rank_2_area: Finds the coordinates of the top left box on the area for which core with rank "r" is responsible

rank_2_row_col: Calculates the position of the area for which core with rank "r" is responsible. For example, if the number of worker cores is 16, then the core with rank 1 is responsible from the area whose position is (0,0). Likewise the core with rank 2 is responsible from the area whose position is (0,1). It is (0,2) for the rank 3 core.

2- COMMUNICATION BETWEEN THE MANAGER AND WORKERS



At the beginning of the program, manager core handles the input operations. When it obtains the whole map, it sends the related area to each worker processor. The responsibility area of each worker processor is decided by dividing the map into [M] grids. You can see on the right side how the sharing is done for [M] is 16 and the map size is 36x36. By the way, the program is implemented in such way that it assumes always the map size is 360x360. After the manager sends the information to the workers, communication between cores starts. When the iterations are done, each worker sends their final local maps to the manager.

3- COMMUNICATION BETWEEN WORKERS

Communication between workers starts after they get their local maps. I will explain the communication process step by step.

1- Each worker core with even row (zero-based indexing) sends its bottom row to the core under it (For the cores on boundaries, you can imagine that the map is like a toroid shape, which means each core has 8 neighbors.)

2- Each worker core with odd row sends its bottom row to the core under it.

3- Each worker core with odd row sends its top row to the core above it.

4- Each worker core with even row sends its top row to the core above it.

5- Each worker core with even column sends its right row to the right neighbor core.

6- Each worker core with odd column sends its right row to the right neighbor core.

7- Each worker core with odd column sends its left row to the left neighbor core.

8- Each worker core with even column sends its left row to the left neighbor core.

Now every core has the necessary information to simulate the game, except the corners. We can say that, every core has the information of the top corners which the core under it needs. Likewise, every core has the information of the bottom

corners which the core above it needs. To transfer this information, each core makes four more transfer as following.

9- Each worker core with even row sends the information which the core under it needs for its top corners.

10- Each worker core with odd row sends the information which the core under it needs for its top corners.

11- Each worker core with odd row sends the information which the core above it needs for its bottom corners.

12- Each worker core with even row sends the information which the core above it needs for its bottom corners.

4- OUTPUT

When the manager receives the all local maps, it produces an output file on which it writes the whole final map. Then the program terminates.

Improvements

First of all, the communication model can be improved, because at each iterations cores transfer information twelve times which can be reduced with more clever way of communication model.

In order to improve the communication model, we may also change the grid-split design with a design that makes possible the better communication models.

Difficulties Encountered

This was my first parallel programming experience, therefore it was difficult for me to get used to this paradigm. Other than that, I did not encounter much difficulties. Deciding the communication model for checkered and periodic design was also challenging.

Conclusion

With this project, I have learnt how I should think when doing parallel programming and how to avoid deadlocks with a good design of communication model. I also get familiar with the working principles of highly parallel computing units, like GPUs.