Project #3 - Conway's Game of Life in MPI

SWE3021 Multicore Computing - Fall 2020

Due date: November 27 (Fri) 17:00pm

1 Goal

The purpose of this assignment is to give you experience writing a message passing program with MPI to building your understanding of message passing.

2 Game of Life

Game of Life is a cellular automaton formulated by the British mathematician, John Orton Conway in 1970. The "game" is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input. The user just sets up the initial configuration and leaves the game to evolve on its own. Conway invented Life as a simplification of a much more complex model by John Von Neumann. Von Neumann had invented his automaton in a search for a hypothetical machine that could build copies of itself, and Conway's Life can do the same. What particularly interests the computational community is that Life is Turing Complete, meaning it can compute any algorithm that a conventional computer can.

3 Details

The Game of Life models some genetic laws for birth, death, and survival. Consider a checkerboard consisting of an n-by-n array of cells. Each cell can be either alive (denoted by #) or dead (denoted by .).

The next state of each cell depends on the states of its neighbors. Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

- Any live cell with fewer than two live neighbors dies, as if caused by under-population.
- Any live cell with two or three live neighbors lives on to the next generation.
- Any live cell with more than three live neighbors dies, as if by over-population.
- Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

The initial pattern constitutes the seed of the system. The first generation is created by applying the above rules simultaneously to every cell in the seed—births and deaths occur simultaneously, and the discrete moment at which this happens is sometimes called a tick (in other words, each generation is a pure function of the preceding one). The rules continue to be applied repeatedly to create further generations.

The earliest interesting patterns in the Game of Life were discovered without the use of computers. The simplest static patterns ("still lifes") and repeating patterns ("oscillators") were discovered while tracking the fates of various small starting configurations using graph paper, blackboards, physical game boards (such as Go) and the like.

For more information about this game, please refer to

`https://en.wikipedia.org/wiki/Conway's_Game_of_Life`

4 Your Program

Your should write a parallel program to simulate Conway's Game of Life.

5 Input

For this project the game board has finite size. The x-axis starts at 0 and ends at X_limit-1 (supplied on the command line). Likewise, the y-axis start at 0 and ends at Y_limit-1 (supplied on the command line).

Your program should read in a file containing the coordinates of the initial cells. Sample files are located in swe3021/gameoflife/life.data.1 and life.data.2. You can also find many other sample patterns on the web (use your favorite search engine on "game of life" and/or "Conway").

Your program should take five command line arguments: the name of the data file, the number of processes to invoke (including the initial one), the number of generations to iterate, X_limit, and Y_limit.

6 Output

Your program should print out one line (containing the x coordinate, a space, and then the y coordinate) for each occupied cell at the end of the last iteration.

7 Hints

Figure out how you will decompose the problem for parallel execution. Please note that MPI does not have great communication performance over 1G Ethernet. So, you will want to make message passing infrequent. Also, you will need to be concerned about load balancing.

One you have decided how to decompose the problem, write the sequential version first.

8 How to Parallelize

Your parallel version must use MPI to parallelize the computation. The grid must be partitioned into small grids, with one (or more) row/column of padding on each side for cells whose values will be computed by other processors.

You can develop this program incrementally. For example,

- First, get each process to do its local computation without any communication.
- Second, start the communication by having processes exchange a single value (for example, each process with a left neighbor sends its upper-left cell, then each process with a right neighbor receives that value)
- If exchanges of a single value work, change to receiving an entire column or row.
- Once one communication (e.g., send left/receive right) is working, add another.

9 How to Run

If this is the first time you compile and run MPI programs in In-Ui-Ye-Ji cluster, you must add the following lines to your `.bash_profile` file.

```
#MPI_HOME
MPI_HOME=/usr/local/openmpi-1.10.0
PATH=$PATH:$MPI_HOME/bin
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MPI_HOME/lib

export PATH
export LD_LIBRARY_PATH
```

Once you added these lines, reload the `.bash_profile` by typing

```
source ~/.bash_profile
```

Now you are ready to compile MPI programs using `mpiCC` compiler.

```
$ mpiCC -o project3 project3.cpp
```

For testing purposes, sample input and output files will be provided in /home/swe3021/project3/. Please copy them to your working directory and run the following `mpirun` command.

```
$ mpirun -np 16 ./project3 < input.txt  > my_output.txt
```

This command will run 16 processes on your local host. To run your program on multiple hosts, you need to use the following command.

```
$ mpirun -hostfile hosts.txt -np 32 -bynode ./project3 < input.txt  > my_output.txt
```

Note: < will read the text file (input.txt) and "redirect" the contents to the standard input so that your program can read the file with `cin` statements. The output of your program will be captured by another IO redirection operator ">" and stored in a file my_output.txt.

```
$ diff -bwi my_output.txt  sample_output.txt
```

The diff command displays any difference between two files. If it does not show any output, your output is correct.

For any question regarding this project, please post in Piazza Q&A so that we can share questions and answers with all other students and TAs. Also, please check Piazza regularly for any updates on this project.

10 Grading

The goal of this project is not to write the most efficient implementation of Game of Life, but rather to learn MPI programming. So, the score of this project will be given as the following.

The project will be graded as follows:
Correctly runs on 1 processor: 10 %
Correctly runs on 36 processors on 3 nodes: 40%
Performance on 1 processor: 10%
Relative performance of parallel version: 40%

11 How to Submit

To submit your project, you must name your code to project3.cpp and run `multicore_submit` command in your project directory in swin.skku.edu server.

```
$ multicore_submit project3 project3.cpp
```

This command will submit your code to the instructor's account.

For any questions, please post them in Piazza so that we can share your questions and answers with other students. Please feel free to raise any issues and post any questions. Also, if you can answer other students' questions, please don't hesitate to post your answer. You would get some credits for posting questions and answers in Piazza.