CSE412 Parallel Computing (Fall 2022)
Instructor: Prof. Myeongjae Jeon
Due date: December 11, 2022, 11:59 pm.

Assignment 2: Conway's Game of Life in MPI

The purpose of this assignment is to give you experience on writing a message passing program with MPI to build your understanding of message passing programming model. The program you will need to write is 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.

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 neighbors, 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

Your job is to write a parallel program to simulate Conway's Game of Life following the steps below:

1. Prompt a user to enter the size of a board (*m*). *m* is an arbitrarily large integer.
2. Prompt a user to enter the number of final generation to display (*N*).
3. Prompt a user to enter how many ghost columns and rows to use.
4. Prompt a user to enter the state of cells, and store the states in a two-dimensional array.
5. Let the master process display the final N'th generation. Note that the generation changes all at once. Only current cells are used to determine the next generation.

How to Parallelize

Your parallel version must use MPI to parallelize the computation. The grid (n by n array of cells) 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.

Ghost Cells

Once the program is working, you should consider making the communication more efficient using so-called “ghost cells”. Ghost cells are cells in boundaries that are assigned to a neighbor process, but they are duplicated for performance reasons. By adding k rows and columns of ghost cells, each process will hold a copy of the data in neighbor processes simply to make the computations and message passing at the process easy. Besides this, we can reduce the number of message passing if we increase the number of rows and columns of ghost cells. Your program will prompt the user to enter the number of columns and rows of ghost cells.

How to Run

Please use Googling results on how to install MPI, including https://docs.open-mpi.org/en/v5.0.x/installing-open-mpi/quickstart.html

Firstly, using WSL or linux-based operating systems to follow the guideline below (WSL, virtual machine (VMware or Virtual Box), Dual Boot). Then, add the following lines to your .bash_profile file in your home directory (thus at ~/.bash_profile).

```
#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 add 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. Please try run the following `mpirun` command, which will run 4 processes on your local host.

```
$ mpirun -np 4 ./project3 < input.txt > my_output.txt
```

(optional) To run your program on multiple hosts, you need to use the following command, where `hosts.txt` includes a list of server IPs (or domain names) in which you are going to run the program.

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

Note that "<" 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.

Submitting your Assignment

You should submit a **assignment2_[studentID].tar.gz** that contains:

- a directory containing the code for your MPI program, and
- a writeup about your programs in either Word or PDF format (PDF preferred).

You will submit your assignment by uploading the compressed file on BlackBoard. If the submission comes with a wrong format or the code will not build, you will not receive points. Your assignment should be submitted prior to the assignment deadline noted at the top of this handout, or it will be subject to the policy on late work. If you have any issue on Blackboard submission, please send an email to the TA (tykim8191@unist.ac.kr).

Grading Criteria

- 20% Program correctness.
- 30% Program scalability and performance.
- 10% Program clarity, elegance and parallelization. The programs should be well-documented internally with comments.
- 40% Writeup. Your grade on the writeup includes your approaches on parallelization, and an analysis on the program performance and scalability. We care about the quality of the writing (grammar, sentence structure), whether all of the required elements are included, and the clarity of your explanations.

Notes

- Make sure your code is compiled and works on Linux.
- Make sure your code is not generating segmentation fault.
- Do not change a given code file name <project3.cpp>.
- Direct discussion with others is strictly prohibited.
- For any questions, please use a piazza.
- For collaborating with others, follow the rules from the slides for the first lecture on Blackboard. Please write down the student who collaborates with you at the first of your report.