Project 6 – Game of Life MPI Implementation

Due: 12/13/2021 11:59:59 pm

Goal. In this project, you will be parallelizing Conway's Game of Life (https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life) in C using the OpenMPI library.

Setting up. For this project, you will need a machine that supports OpenMPI and the Make command. Follow the instructions below for your OS

Linux. Install MPI on your machine by calling:

- `run/install_mpi.sh`

Typically, UNIX based machines will have `make` pre-installed. If this is not the case, you can install it by calling:

- `sudo apt install build-essential`

Mac. Install Homebrew by following the link here: https://brew.sh/. Then, ensure you have XCode Command Line Tool (CLT) installed by calling:

- `sudo xcode-select --install`

Upon installation of brew and XCode CLT, you can install MPI and Make by calling:

- `brew install open-mpi`
- `brew install make`

Windows. We recommend you enabled Windows Subsystem for Linux (WSL) for this project. Follow the instructions below:

- Go to "Control Panel" -> "Programs" -> "Turn Windows features on or off"
- Check "Windows Subsystem for Linux" and click "OK"
- Restart your device. Upon restart, open the Microsoft Store
- Search "Ubuntu" and install the "Ubuntu" app
  - "Ubuntu20.04 LTS" and "Ubuntu18.04 LTS" should work if you already have them pre-installed
- Launch the Ubuntu app. If prompted, create an account

From WSL, you may access files on your Windows machine through `/mnt/c/.` This directory is the same as Windows (C:). Navigate to the P6 directory by calling:

- `cd /mnt/c/<Windows-path-to-p6-directory>`

Now let's install `make` and OpenMPI. Run the following commands from the P6 directory within Ubuntu WSL:

- `sudo apt update`
- `sudo apt install make`
- `run/install_mpi.sh`

Getting started. We have provided you with many files to the program, which we will describe below:

- A serial (sequential) version of the program called `src/life_seq.c`. Take a look at how the game is implemented. You may use code from this file in your MPI implementation. The serial version will also be used to generate the canonical output for the program. You should feed the input files into this program in order to see what the correct output is.
- The starter code for the program called `src/life_mpi.c` **This is the only file you will be modifying in this project.**
- A sample MPI program that shows how message passing works called `src/sample_mpi.c`
- In order to compile all of the examples and the `src/life_mpi.c` programs, you should go to the `run/` directory. You can either use the `Makefile` or the `run_*.sh` scripts to build and run your programs.
- Visualization and random input generation scripts are performed by `tools/life_visualizer.py` and `tools/input_generator.py` respectively. There is an attached `tools/TOOLS_README.txt` document with these two files which explains how they function.
- For more information, please see the `README` file under the project directory.

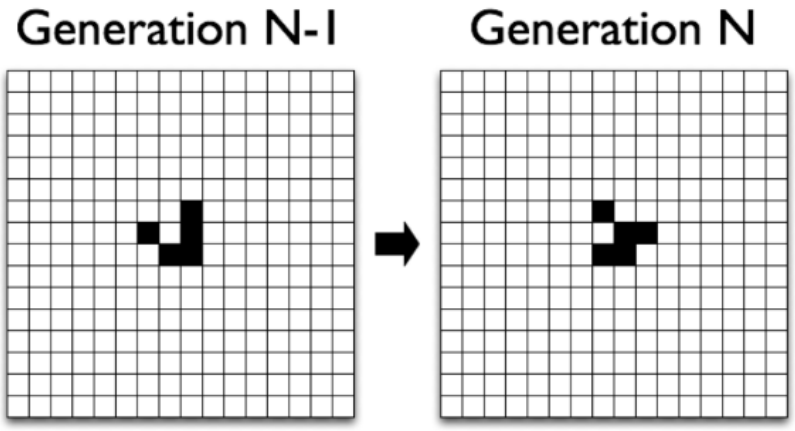
How the game works. The Game of Life consists of an N by M matrix, where each cell can either be 0 (dead) or 1 (alive). The game will start with the cells corresponding to the (x, y) coordinates in the initial input file being set to 1 (alive). This is the “first generation” of the game. The game consists of constructing successive generations of the board based on certain rules which are described below:

1. Cells with 0,1,4,5,6,7, or 8 neighbors die (0,1 of loneliness and 4-8 of overpopulation)
2. Cells with 2 or 3 neighbors survive to the next generation
3. An unoccupied cell with 3 neighbors becomes occupied in the next generation.

In `life_seq.c`, two copies of the board are defined: *`prev`* to store the previous generation and *`curr`* to compute the next generation. Boards are padded on all sides to allow for easier computation (there are less if-statements when computing neighbors of a cell on the board’s edge).

While you are not required to, we recommend you follow this paradigm when implementing Game of Life with MPI.

An example graphic of one iteration of the game is found below:

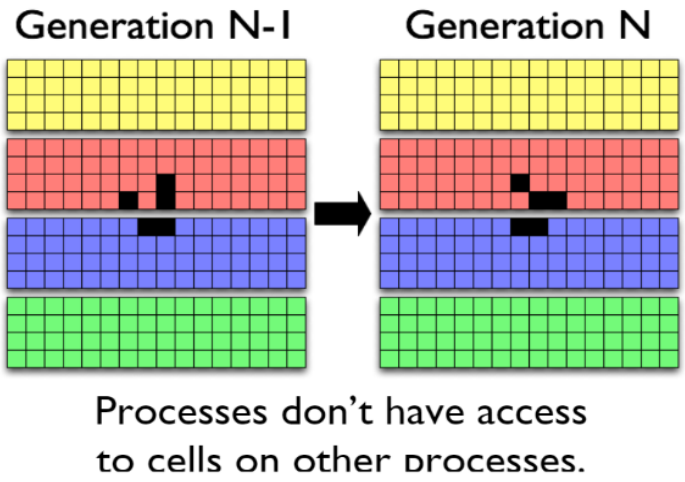


For more information on the Game of Life, check out the link at the top of the pdf.

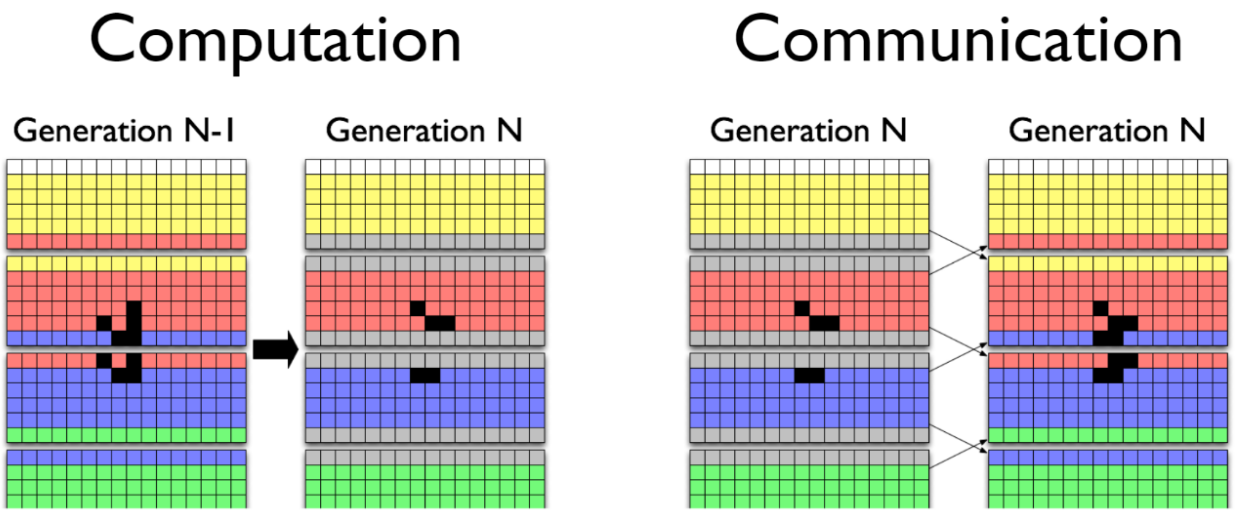
Implementing the game using MPI. We have provided some starter code, which includes all of the initialization of the MPI functionality for the program, as well as the code for the rank 0 process. **You will need to implement the code for the rank 1 to n processes (worker processes).**

The following is a high-level overview of this implementation:

1. Upon start-up, the rank 0 process will partition the board into non-overlapping sub-boards and assign them to each worker process.
    a. Worker processes will allocate memory to create their sub-board
    b. Worker processes do not have direct access to each other's boards
2. Worker processes are responsible for computing up to the *N-th* generation of the Game of Life for their sub-board. *N* is specified as a runtime parameter.



   a. To compute the next generation of its top/bottom rows, the worker process needs to know the top/bottom rows of its neighboring processes
3. Using MPI, worker processes will communicate their top and bottom rows to their respective neighbors. These rows will be stored in the sub-board's padding
4. With the updated sub-board, each worker process will compute the next generation of its sub-board. This logic should be the same as in `life_seq.c`



5. After *N* iterations of this cycle, all worker processes will send their final sub-board state back to the rank 0 process.
6. The rank 0 process collects these sub-board states into one large board. It then prints out the overall *N-th* generation in the Game of Life.

The BOARD macro: Conceptually, you can think of boards as 2-D matrices with padding on all sides: they are of size *(height+2)*(width+2)*. However, they are stored in C as a very long 1-D array of integers. We define the `BOARD(board, x, y)` macro to

help you easily access indices of the 1-D array **as if it were a 2-D matrix, centered around the original unpadded board.**

With *BOARD*, the padded rows/columns are now at indices *-1* and *height/width*. The top left most colored cell can now be imagined as the 0th column, 0th row.

For example:

- *BOARD(board, -1, -1)* returns the value of the top-left in the padded board
- *BOARD(board, 0, 1)* returns the value at the 1st column, 2nd row in the padded board (this is the same cell as the 0th column, 1st row in the unpadded board).

We highly recommend you use the *BOARD* macro to simplify computation in your implementation.

Running the serial and MPI programs. First, go to the `run/` directory. Then, you can use either the `Makefile` or the `run_*.sh` files to run the programs. Take careful note of the runtime parameters you must supply the `run_*.sh` files. *Note: you MUST run the* `run_*.sh` *files from within the* `run/` *directory*.

On some machines, running the MPI Game of Life may output the following lines. You can ignore these messages as they do not affect the project's execution:

- "WARNING: Linux kernel CMA support was requested…"
- "1 more process was sent help message …"
- "Set MCA parameter "orte_base_help_aggregate" to 0 to see all help …"
- "A system call has failed during shared memory initialization… Error: No such file or directory (errno 2)"

Testing with the sample input/output. You can use the 3 sets of sample input/output files located in the `run/` directory to test your program.

The names of 3 sample output files are formatted as follows:

```
life-data_<sample #>_<# of generations>-<x limit>-<y limit>.out
```

Testing with the input generator tool. First, you can create random inputs for the program using the `input_generator.py` script. Next, you should run these input files either through the visualizer or the serial version in order to get the correct output for the final generation that you specified. Finally, run the MPI program with the input file and get the output from that.

Install the dependencies of the the `.py` files in `tools/` by calling:

- `sudo apt upgrade`
- `sudo apt install python3`

- `sudo apt install python3-pip`
- `pip3 install numpy`
- `pip3 install matplotlib`

The following dependencies are enough to **generate** input and sequential output files using `input_generator.py` and `life_visualizer.py`, respectively.

Windows: **The following instructions are completely optional and only useful to visualize the sequential game of life within WSL.**

We will need an X11 server for visualization. Download VcXsrv by visiting https://sourceforge.net/projects/vcxsrv/. After installation, launch xlaunch.exe to configure your display settings and start up VcXsrv

Within Ubuntu WSL, run the following commands

- `sudo apt install python3-tk`
- `export DISPLAY=localhost:0.0`

You are now able to visualize matplotlib animations when calling `life_visualizer.py`

Comparing the output files. We recommend using the `diff` command (or https://www.diffchecker.com/ if you prefer a graphical display) to verify that the output from your MPI program is the same as the output from the serial version.

Submission. Submit the `life_mpi.c` file to Gradescope.

Grading. Since you can easily test your program locally on random input, all tests on our side will be secret.