CSCI-4320/6360 - Assignment 1: Conway's Game of Life Using 1-D Arrays

Christopher D. Carothers
Department of Computer Science
Rensselaer Polytechnic Institute
110 8th Street
Troy, New York U.S.A. 12180-3590

January 24, 2020

DUE DATE: 11:59 p.m., Friday, January 31st, 2019

1 Overview

To prepare you for upcoming programming assignments in CUDA, you are to construct a C program that specifically implements Conway's *Game of Life* but with the twist of using ****only**** 1-D arrays. Additionally, you will run this C-program on the *AiMOS* supercomputer at the CCI in serial mode. This will prepare you for the batch queuing system of CCI computing resources.

1.1 Game of Life Specification

The Game of Life is an example of a Cellular Automata where universe is a two-dimensional orthogonal grid of square cells (with WRAP AROUND FOR THIS ASSIGNMENT), each of which is in one of two possible states, *ALIVE* or *DEAD*. 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 at each and every cell:

- 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 world size and initial pattern are determined by an arguments to your program. A template for your program will be provide and more details are below. The first generation is created by applying the above rules to every cell in the seedbirths and deaths occur simultaneously, and the discrete moment at which this happens is sometimes called a "tick"

or iteration. The rules continue to be applied repeatedly to create further generations. The number of generations will also be an argument to your program. Note, an iteration starts with $Cell(0,0)$ and ends with $Cell(N-1, N-1)$ in the serial case.

1.2 Template and Implementation Details

For the provided template, *gol.c*, there are the following functions you must write.

- **gol_swap**: this function will perform the swap operation of the two pointers.
- **gol_countAliveCells**: this function check each of the 8 neighbors and counts the alive cells. Note, you will have to access the `data` array by performing you own pointer math using the `x0, x1, x2, y0, y1` and `y2` arguments.
- **gol_iterateSerial**: this function takes as an argument the number of iterations and compute the world and swaps the new world with the previous world to be ready for next iteration. There are for-loops already defined. You need to fill in the code in the for-loops.

To compile and execute the GOL program, do:

- compile: `make all` – this will invoke the GCC compiler with debug and all warnings turned on.
- run: `./gol 4 32 2` – this will execute the GOL program using pattern 4 for a world size of 32x32 and perform two iterations. Pattern 4 is a "spinner" pattern that is at the corners of the grid. For this assignment, all world sizes will be 32x32.

For this assignment, there are the following 5 patterns:

- Pattern 0: World is ALL zeros.
- Pattern 1: World is ALL ones.
- Pattern 2: Streak of 10 ones in about the middle of the World.
- Pattern 3: Ones at the corners of the World.
- Pattern 4: "Spinner" pattern at corners of the World.

1.3 Helper Information

To aide you the development of the above functions and 2-D array indexing using only a 1-D array data structure, please use the following algorithmic help.

First, if you had a 2-D array structures, and you wish to know if `data[x1][y1]` is alive or dead, you would need to know the status of cells `data[x1-1][y1-1]`, `data[x1+1][y1-1]`, `data[x1+1][y1-1]`, `data[x1-1][y1]`, `data[x1+1][y1]`, `data[x1-1][y1]`, `data[x1-1][y1+1]`, `data[x1][y1+1]` and `data[x1+1][y1+1]`. However, we need to access to be for a 1-D array. So, let's compute the pointer address offsets need. First, we want `data[x1][y1]` to become `data[x1 + y1]` and we need to determine what `x0` (e.g., `&data[x1 -1]`) and `x2` or `&data[x1 + 1]` will be as well as `y0, y2` will be be. Also, we need to account for the factor that the 2-d world has wrap-around. For this, the x and y pointer offsets become:

```
y0 = ((y + g_worldHeight - 1) % g_worldHeight) * g_worldWidth;
y1 = y * g_worldWidth;
y2 = ((y + 1) % g_worldHeight) * g_worldWidth;
```

And for an x values over the range of 0 to `g_worldWidth-1`:

```
x1 = x
x0 = (x1 + g_worldWidth - 1) % g_worldWidth;
x2 = (x1 + 1) % g_worldWidth;
```

2 Running on AiMOS

Please follow the steps below:

1. Login to CCI landing pad (`lp01.ccni.rpi.edu`) using SSH and your CCI account and password information. For example, `ssh SPNRcaro@lp03.ccni.rpi.edu` and at prompt type in password.
2. Login to *AiMOS* front end by doing `ssh PCP9yourlogin@dcsfen01`.
3. (Do one item only). Setup ssh-keys for passwordless login between compute nodes via `ssh-keygen -t rsa` and then `cp /.ssh/id_rsa.pub /.ssh/authorized_keys`.
4. Load modules: run the `module load gcc` command. This puts the GNU C compiler in your path correctly as well as all needed libraries, etc.
5. Compile code on front end by issuing the `make` command.
6. Get a single node allocation by issuing: `salloc -N 1 -t 30` which will allocate a single compute node for 30 mins. The max time for the class is 1 hour per job. Your `salloc` command will return once you've been granted a node. Normally, it's been immediate but if the system is full of jobs you may have to wait for some period of time.
7. Use the `squeue` to find the `dcsXYZ` node name (e.g., `dcs24`).
8. SSH to that compute node, for example, `ssh dcs24`. You should be at a Linux prompt on that compute node.
9. Issue run command for GOL. For example, `./gol 4 32 2` which will run GOL using pattern 4 with a world size of 32x32 for 2 iterations.
10. If you are done with a node early, please `exit` the node and cancel the job with `scancel JOBID` where the JOBID can be found using the `squeue` command.

3 HAND-IN and GRADING INSTRUCTIONS

Please submit your C-code to the `submitty.cs.rpi.edu` grading system. The grading rubric will be available to test your solution. Please make sure you document the code you write for this assignment. That is, say what you are doing and why.