Parallel Computing/Programming Assignment #2: Parallel Game of Life

Christopher D. Carothers
Department of Computer Science
Rensselaer Polytechnic Institute
110 8th Street
Troy, New York U.S.A. 12180-3590
Email: chrisc@cs.rpi.edu

February 12, 2016

DUE DATE: Noon, Friday, February 26th, 2016

1 Assignment Description

For this assignment, your are use the serial Conway's Game of Life program you wrote in assignment 1 and develop a parallel version and leverage the RNG provided in the template.

1.1 Review: Basic Rules

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 initial pattern constitutes the seed of the system using a Random Number Generator (RNG) at time 0. Use the RNG provided in the assignment 2 code template (yes, a new template!). 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" The rules continue to be applied repeatedly to create further generations.

Note, a "tick" starts with $Cell(0,0)$ and ends with $Cell(N-1,N-1)$ in the serial case. When we get to parallel things will get much more interesting! (see below).

1.2 Adding Additional Randomness

Using the uniform distribution provided in RNG (see the template on how to use this new RNG), if value return by the RNG is greater than a set threshold, then perform the above described basic rules for each cell. Otherwise, randomly pick state of *LIVE* or *DEAD*.

1.3 Parallelization Approach

First, this time to ensure that there is more true "randomness" among the cells, each row will have it's own RNG seed set. According to the template, every MPI Rank will initialize 16,384 RNG streams. These streams are about 2^{70} calls apart.

For the parallelization approach, each MPI Rank will perform an even "chunk" of rows for the Cellular Automata universe. For example, suppose you have a 1024x1024 universe using 16 MPI Ranks. If you divide the universe equally, each Rank will have 64 rows or 64x1024 cells to compute. Thus, Rank 0, will compute rows 0 to 63, Rank 1 computes rows 64 to 127 and Rank 2 will compute rows 128 to 191 and so on.

For the Cellular Automata universe allocation each MPI Rank only needs to allocate it's specific chunk plus space for "ghost" rows at the MPI Rank boundaries. The "ghost" rows can be held outside of the main MPI Rank's Cellular Automata universe.

Now, you'll notice that rows at the boundaries of MPI Ranks need to be updated / exchanged prior to the start of computing each tick. For example, using the above 1024x1024 universe and 16 MPI Ranks example, MPI Rank 0 will need to send the state of row 0 (all 1024 entries) to MPI Rank 15. Additionally, Rank 15 will need to send row 1023 to Rank 0. Also, Rank 0 and Rank 1 will do a similar exchange.

For these row exchanges, you will use the `MPI_Isend` and `MPI_Irecv` messages. You are free to design your own approach to "tags" and how you use these routines except your design should not deadlock.

So the algorithm becomes:

```
main(...)
   {
      Setup MPI (template has this already!)
      if Rank 0, start time with MPI_Wtime.
      Allocate My Rank's chunk of the universe +
         space for "ghost" rows.
      Randomly initalize my universe using the
         right RNG stream for each row.

      for( i = 0; i < number of ticks; i++)
         {
            Exchange row data with MPI Ranks
               using MPI_Isend/Irecv.
```

```
            Do rest of universe update as done
               in assignment 1 except use the
               correct row RNG stream and use the right
               "ghost" row data at Rank boundaries
         }

      MPI_Barrier();
      if Rank 0, end time with MPI_Wtime.

      Output performance results or
         My Rank's chunk of universe to separate file.

      MPI_Finalize();
   }
```

A final note: your program is NOT parallel deterministic. That means that your Cellular Automata universe will differ at each step/tick in the computation depending on how many MPI Ranks you use. This is because the exchange of rows is using data computed in a prior tick and does not get the benefit of the update dependencies that happen within a "tick".

1.4 Experiments

There are two series of experiments you will conduct using the class server, `kratos.cs.rpi.edu`.

- Run a 128x128 cell universe using 16 MPI Ranks for 100 ticks with a threshold of 0%.
- Run a 128x128 cell universe using 16 MPI Ranks for 100 ticks with a threshold of 25%.
- Run a 128x128 cell universe using 16 MPI Ranks for 100 ticks with a threshold of 50%.
- Run a 128x128 cell universe using 16 MPI Ranks for 100 ticks with a threshold of 75%.
- Run a 128x128 cell universe using 16 MPI Ranks for 100 ticks with a threshold of 90%.

Here, each MPI Rank will output will open a file using `fopen` with a distinct name (suggest appending the Rank number to the file name) and output it's data in either a binary or text format, depending on your approach to graphing these results.

You will need to make sure you "glue" the results files together in the right order.

Plot each of these experiments in a cell graph (yes, you'll have 2048 data points this time!). In your write-up describe how these plots differ and how they compared with your previous smaller 16x16 plots.

In the second series of experiments you will capture the execution time by using `MPI_Wtime()`. In example code for this would be:

```
{
   double starttime, endtime;
   starttime = MPI_Wtime();
    ....  stuff to be timed  ...
```

```
    endtime   = MPI_Wtime();
    printf("That took %f seconds\n",endtime-starttime);
}
```

Start your timer on Rank 0 just after `MPI_Init` and stop the timer just before `MPI_Finalize`. Then print out the execution time however do not print the output for this series of experiments. This series is measuring the "strong scaling" parallel performance across the following set of runs:

- Run a 1024x1024 cell universe using 1 MPI Rank for 100 ticks with a threshold of 25%.
- Run a 1024x1024 cell universe using 2 MPI Rank for 100 ticks with a threshold of 25%.
- Run a 1024x1024 cell universe using 4 MPI Rank for 100 ticks with a threshold of 25%.
- Run a 1024x1024 cell universe using 8 MPI Rank for 100 ticks with a threshold of 25%.
- Run a 1024x1024 cell universe using 16 MPI Rank for 100 ticks with a threshold of 25%.

Plot execution time (Y-axis) as a function of the number of MPI Ranks (X-axis). Also, in your report, discuss the max speedup relative to the execution time for the serial case you obtain as well as at what point is your parallel efficiency the greatest (i.e., speedup / MPI Ranks for each case and which case yields the highest value).

1.5 Assignment 1 Code Template

Is available on `kratos.cs.rpi.edu` under `/home/chrisc/assignment2-template.tar.gz`. To untar it to your home directory, do the following:

- `zcat /home/chrisc/assignment2-template.tar.gz | tar -x`.

That will create an `template` sub-directory. Rename that sub-directory by doing `mv templateassignment2`. Inside the `assignment2` sub-directory you will see the code files which will be covered in class.

YES, YOU MUST USE THE TEMPLATE FOR THIS ASSIGNMENT. FAILURE TO DO SO WILL RESULT IN A SIGNIFICANT GRADE PENALTY.

2 HAND-IN INSTRUCTIONS

Keep/put your assignment C code and write-up with graphs in PDF format on the class server, `kratos.cs.rpi.edu` under the `assignment2` sub-directory on your Linux account.

We will use an automated script to collect the assignments. So it is extremely important you place your solution in the right directory name, `assignment2`.

PLEASE PUT YOUR ASSIGNMENT UNDER THE `assignment2` SUB-DIRECTORY OR ELSE IT WILL NOT BE COLLECTED CORRECTLY AND COULD RESULT IN SUBSTANTIAL POINT LOSS OR NOT GRADED AT ALL.

ADDITIONALLY, KEEP A BACKUP COPY OF YOUR CODE, DATA and REPORT/GRAPHS ON YOUR OWN SYSTEM IN CASE KRATOS.CS.RPI.EDU FAILS DURING THE ASSIGNMENT PERIOD.