CSC 4760 Philip Westrich
Assignment #4 Monday, October, 24, 2015

For this assignment, we were to simulate the Game of Life problem using a square, wraparound grid of size 40,000 using OpenMP. We were to run it for 1, 2, 4, 8, 16, and 32 threads and 100, 200, and 400 generations, record runtimes, calculate theoretical and actual speedup and efficiency, and graph the results. As a bonus part of the assignment, we were also to submit an animation of the generations progressing for a smaller problem size.

Each cell of the grid in this problem is randomly initialized to one of two values: alive or dead. For each generation, a cell dies from either overpopulation (more than three adjacent alive cells) or starvation (fewer than two adjacent cells), is born again from reproduction (exactly three adjacent live cells), or continues to live (is alive and has two or three live neighbors).

All of this takes place simultaneously each discrete time step, or generation. Because each cell's state in the next generation is dependent on only it's adjacent cell's states in the current generation, each cell's state for the next generation can be calculated independently of every other one for this generation. Each generation must be calculated sequentially, however. Therefore, I decided to, for each generation, divide the number of cells evenly using OpenMP's static assignment, calculate the next generation for each cell, wait for each thread to finish, then move on to the next generation.

As it can be seen from the attached graphs, speedup increases as the number of threads increases, and both efficiency and execution time decrease as the number of threads increases. Theoretical speedup was also calculated for every run. In a few cases, especially in fewer generations, the actual speedup was greater than the calculated theoretical speedup. This is likely explained by the graph of setup time vs. thread count; while each program of the same problem size must create and initialize an array of the exact same size in the exact same fashion, the time it takes to do this varies by almost a minute. This is probably due to factors outside our control, such as cache misses, other processes running on the machine, the scheduling of the multiple tasks by the operating system, and such.

To create the visualization of the process, I used a program provided to us that turned text files of the current state of the grid each tick, and turned them into images. To make it easier to view, I generated a size 20 grid for 20 generations, and visualized that. It was not difficult to modify the program to do this; after every generation, I simply wrote to a text file a 1 for a live cell, and a 0 for all dead cells. After running the program, I passed it to the matrix rasterizer we were given, and was gone. The concatenated .GIF file was create with the website that Luke showed us in class.