Introduction
The purpose of this homework assignment is to implement and analyze a parallel version of the Game of Life.

I implemented the Game of Life using Pthreads, distributing the data across threads by groups of rows, i.e. when threads are numbered 0 to NUM_THREADS-1 and there are NUM_ROWS rows, the ith thread is given rows i*NUM_ROWS/NUM_THREADS to (i+1)*NUM_ROWS/NUM_THREADS-1. The threads synchronize via a barrier to insure that all threads are simulating the same generation at the same time.

To test the implementation and the performance of the parallel code with up to 8 processors, I execute the code on the Natural Sciences Compute Cluster (NSCC) at Colby. Each run uses a similar board tiled with copies of the Game of Life oscillator known as the toad. For each generation, each cell must be computed once. Thus we measure the problem size N in terms of cellular computations, i.e. N = 64*BOARD_SIZE*NUM_GENERATIONS where NUM_GENERATIONS is the number of generations over which the game evolves, BOARD_SIZE is the number of tiles on the board, and each tile contains 64 cells. To compare the influence of board size and generation number on the ability of a parallel solution to improve performance, I hold N constant while varying the number of cells on the board and NUM_GENERATIONS.

I use the C time library function time() to report the runtime in seconds of the sequential and parallel implementations. The runtimes reported for parallel code include the initialization and finalization code for the Pthreads library functions and structs.

Results
Keeping the total problem size constant at N=64*9,000,000, I collected runtimes for 3 board sizes using both the sequential and the parallel implementations. For the parallel implementation, I ran each computational experiment three times and reporting the mean runtime. Figure 1 shows runtimes for Experiment 1 in which a 1000x1000 tile board is simulated for only 9 generations. Figure 2 shows runtimes for Experiment 2 in which a 3x3 tile board is simulated for 1,000,000 generations, and Figure 3 shows runtimes for Experiment 3 in which a 30x30 tile board is simulated for 10,000 generations.
Figure 1. Results from Experiment 1 include runtimes for simulations of a 1000 tile by 1000 tile board over 9 generations. For the parallel algorithm, each point represents the mean runtime (in seconds) for 3 trials.

Figure 2. Results from Experiment 2 include runtimes for simulations of a 3 tile by 3 tile board over 1,000,000 generations. For the parallel algorithm, each point represents the mean runtime (in seconds) for 3 trials.
Discussion

The data show clearly that multiple processors speed up the simulation only when the board is sufficiently large – only in Experiment 1 does increasing the number of processors always decrease the runtime. The effect of multiple processors varies markedly across the experiments. For the smallest board (Experiment 2), increasing the number of processors always decreases the speed of the code (see Figure 2). This is not due to any inherent difference between the three problems – the runtimes for the sequential implementation are consistent (at 50 seconds) across all experiments. The differences in runtimes must be caused by overhead in the parallel implementation.

An obvious candidate for significant parallel overhead is the barrier. The barrier synchronizes all threads every generation. Thus, the barrier is called over 100,000 times more in Experiment 2 than in Experiment 1 and it is not surprising that the runtimes in Figure 2 are significantly greater than those in Figure 1. However, the number of barrier calls alone fails to explain the barrier-induced slowdown. In Experiment 2, all experiments use the same number of barrier calls, but increasing the number of processors slows execution. This demonstrates that barrier code takes longer to execute when more threads must reach it, partly due to coordination of more threads, and partly due to the fact that not all threads reach the barrier at the same time. Further, in Experiment 2 the board is small enough that the ratio of each thread’s cellular computation time to barrier code time is small. Thus these data indicate that barrier code dominates execution time.

For Experiment 1, the barrier is executed so infrequently that its effects are negligible. In fact, doubling the number of processors nearly halves the runtimes. Given the coarse granularity of our measurements, the speed-up is indistinguishable from the ideal. We expect that more precise measurements would reveal less than ideal speed-ups.

Experiment 3 straddles a fence – increasing the number of processors from one to two or four improves performance, but increasing further to eight processors degrades performance. This provides additional evidence that the ratio of cellular computation time to barrier execution time must be sufficiently large for multiple processors to be beneficial. As expected, the speedup gained by two and four processors is not as significant in this experiment as in Experiment 1.
In conclusion, the Game of Life can be played with a nearly ideal parallel speed-up, but only when the board is sufficiently large and the number of generations is sufficiently small.

Advice About Write-Ups

- Prize clarity above all else! Do not be so informal as to lack precision, but do not attempt to be so formal that you lose clarity. Remember, the goal of a write-up is to communicate your results and analysis – not to win a contest for using sophisticated words.
- Include an Introduction, Results, and Discussion. The introduction should briefly state the problem and solution. The results section should include the figures and a description of the experiments used to collect the data. All commentary about the data should be left for the discussion section. The discussion section should include in depth analysis of the data and a brief conclusion.
- You may assume that the professor and your classmates are the audience for the paper. This means the introduction can be brief. Use it to provide coherence for the write-up and to explain any terms you introduce (for example, I use BOARD_SIZE for the write-up instead of TILES because the explanation is clearer with the board size).
- Use an active voice and do not be afraid of the first person. For class, it is perfectly acceptable to use “I”, but for more formal work, “we” is appropriate.
- Be concise. Use enough words to make your point clearly, but not so many that the reader becomes bogged down in verbiage.
- Label the axes in your graphs.
- In general, sparse experimental data should be plotted without any line-fitting (and some would even argue there should be no lines connecting the data points). We want to be careful not to impose any function on the data that is not there. For example, when the number of processors is on the x-axis, we should not have a line implying there is data for 1.5 processors. There is no such thing as ½ a processor.
- If you refer to any published papers or books, be sure to cite them.