Everard Francis: 997412811
Calvin Fernandes: 997588873

ECE454: Homework5 Report

Parallelization

The sequential game of life procedure was parallelized using pthreads. Each thread works on a horizontal segment of the board. All segments are equal size.

Thread 1
Thread 2
Thread 3
Thread 4

A visual depiction of how each thread computes its slice on the board.

Synchronization

The barrier primitive was used in order to synchronize all 4 threads. Threads operate on the board on a per iteration basis. At the end of each iteration, a barrier_wait statement is placed in order to make sure that all threads have reached that point before moving on to the next iteration of the loop.

Optimization

Besides parallelizing the code, a number of optimizations were made:

1. Changed the order of the row and column loops in order to optimize the way the cache was accessed. This would result in row major access and an increased amount of cache hits
2. The mod function was replaced by a simple conditional if statements. We did this because the mod function is only useful for the wrap around case which happens twice per line (2 out of 1024 iterations), therefore it should be simplified.
3. The BOARD macro was replaced in order to take advantage of loop invariant code motion that could be performed.
4. The alivep function was optimized and replaced with a simple if-else that was put directly in the loop. Additionally, the else statement from the optimized alivep was removed since GCC failed to optimize this automatically
5. The neighbor count computation was reduced by eliminating computations done by the previous neighboring block. There is a detailed explanation in the comments of the code.
6. A lot of loop invariant code motion code was implemented due to the implementations of the above optimizations further increasing the performance of the code.

7. The O3 optimization flag was used along with the –funroll-loops parameter to allow the compiler to perform more aggressive optimizations, function inlining and loop unrolling.

Detailed Summary of Optimization Results:

Optimization 1: Adding threads. Horizontally divide up the board

1k: Time 45.66, Speedup 4.380201489268 51

Optimization 2: Flipping loops (cache optimization). Horizontally move through the board, by row

1k: Time 35.37, Speedup 5.65450947130337

Optimization 3: Optimizing mod function by simplifying it.

1k: Time 23.73, Speedup 8.42815002107037

Optimization 4: Optimizing BOARD with 1 less pointer access and LICM:

1k: Time 23.23, Speedup 8.6095566078347

Optimization 5: Optimizing alivep by replacing it with simple if-else:

1k: Time 22.95, Speedup 8.71459694989107

Optimization 6: Reducing computation of neighbor_count, by eliminating duplicate computation

1k: Time 22.27, Speedup 8.98

Optimization 7: Moved inner loop to an inline function which helped the compiler expose optimizations

1k: Time 21.98, Speedup 9.10

Optimization 8: Removing else from alivep. Gcc didn't optimize this.

1k: Time 20.79, Speedup 9.62