Team Information

Name	Student Number	UTORId	Email
Connor Smith	1000421411	smithc63	connor.smith@mail.utoronto.ca
Fan Guo	1000626539	guofan1	cfan.guo@mail.utoronto.ca

Our solution accomplishes approximately 60 times speedup compared to the sequential solution using the following optimizations:

1. We use a read board and write board to avoid synchronization overhead. After each generation, these boards are swapped and the new write board is zero'd.
2. We track neighbor counts to simplify transition logic, completely resetting counts between generations and only incrementing the neighbors of live cells. This relates the amount of processing time for each generation to the number of live cells, which generally reduces between generations.
3. We use a look-up table to determine whether a cell should die, stay alive, or become alive between iterations. This table is indexed by the previous neighbor count and whether the cell was alive in the previous generation.
4. We spawn 8 threads, and assign each of them one eighth of the board in a row-wise block order. We do not attempt to load balance threads.
5. Each thread always calculates the first two rows of its block before encountering a pthread barrier. This is to reduce double-write conflicts.
6. Each thread encounters a second pthread barrier after completing its assigned block so the generation transitions are synchronized between all threads.
7. We modified the save_board_values and boards_equalp methods to remove neighbor counts when verifying or exporting results. The verify methods are not encountered in benchmarks, but were modified to ensure the gol_verify binary remains useful. We chose to strip this neighbor info here as it is only necessary in non-performance sensitive use cases (gol_verify) or in places where we must sequentially iterate over the entire board anyway (save_board_values).