**Ting-Hao (Tim) Cheng 997289090**
**Corey Yen 999830090**

# ECE454 HW5 Report

We optimized the Game of Life by using multithreading and various techniques, including a way to significantly reduce the number of reads and writes required to compute each generation.

First, we parallelized the program with multithreading. We chose to create 4 threads, and each thread will be given a quarter of the board to compute. For example, if we have a board of size 64x64, Thread 1 will be in charge of rows 0-15; Thread 2 will be in charge of rows 16-31; Thread 3 will be in charge of rows 32-47; Thread 4 will be in charge of rows 48-63. We chose to use 4 threads because the UG machines has the same number of cores. Therefore by using 4 threads, we can utilize all 4 cores and yield the best performance boost. We used mutex locks, conditional variables, and barriers to control program flow and ensure correctness.

We further optimized our code by repositioning loop motion invariant code; unrolling simple loops; and using local variables instead of referencing pointers. We also took advantage of the static inline keyword, which reduces function calls/jumps.

Lastly, we modified the way to determine a cell’s new state. Instead of scanning all eight neighbours’ states for every cell, we stored all the information (state of the cell and neighbour counts) in a byte. This improved our performance because we only need to check the cell itself to determine whether it should be dead or alive, since the information of all its neighbours are already contained. Furthermore, we would avoid scanning neighbouring cells if no action needs to be done on the cell, and this helps us avoid unnecessary reads and writes.

In conclusion, our attempt of optimizing the Game of Life has provided us with a performance boost of 15.8x. (average time = 12.61 seconds while bench time = 199.95 seconds)