Anthony Alayo 997487401
Freddy Chen   997363124

ECE454 HW5 Report

We optimized the Game of Life by using threads, tiling, and a change in the way that the algorithm checks for its neighbour's state.

The first optimization that we implemented, threading, gave us a performance boost of around 2-3x over the original implementation.We implemented the threading approach by creating 4 threads and each giving them an equal portion of cell states to calculate. We chose 4 as the number of threads because that is the number of cores on the UG machines, thus giving us the best performance utilizing each core fully.

Next we implemented tiling for each thread, to take advantage of caching. Adding this to our threaded implementation didn't give us much performance gain.

Lastly, we changed the way we check for a neighbour's state. In the original implementation we checked all 8 neighbour's states, per cell, per generation, and using that information along with the cell's state, we were able to determine if we should stay alive or die.

In the new implementation, we store a cell's state along with all of it's neighbour's states in a single byte. The motivation behind this is that we can just look at a cell's own value to determine if it should be alive, where we previously checked all neighbour's value as well as a cell's own value.

The overhead needed to maintain the neighbour counts within each byte is the same as the original implementation only if a cell changes state, as we have to inform all of our neighbours of our change. However, in the Game of Life, cells on average don't change. Therefore we no longer pay the overhead of counting neighbouring cells when no change has occurred. This implementation gave us a our biggest performance boost, making our final implementation 13x faster than the orignal.