# ECE 454 Lab 5 Report

Yuan Feng 999284876
Mingqi Hou 999767676

## Optimization: Reducing data access

The performance could be significantly improved by reducing the data access operations in for loops. To achieve this goal, a new data structure array, named *elementNode*, is adopted. *elementNode* has 2 data fields. One of them represents the alive status of the current node. The other one represents the number of neighbors the current node has. In this structure, data is stored in byte using bitfield mechanism, allowing minimum data access. Benefiting from this optimization, the program suffers less from cache misses and explores more spatial locality

In addition, the cell structure is modified to store the alive statues of its neighbours. This modification significantly reduces data access operations and improves cache hit rate. Instead of checking the node’s neighbour (up to 8) for alive status, the alive status of current node can be determined by checking the values stored in the current node.

## Optimization: Reducing unnecessary function calls

In the original program, *Mod()* function is called for all calculations of variables Inorth, isouth, jwest, and jeast. However, this function is only needed for boundary cases. As a result, a if statement is added to determine whether *Mod()* function need to be called.

## Parallelization

The number of threads used is determined by the number of processor cores. 4 threads are used since the processor has 4 cores. For every GOL iteration, the work is split into 4 sub task blocks by row number. For example, if the world size is 32 by 32, each sub task block will have a size of  8 by 32. The boundary rows, which will be shared by different threads are calculated first in sequential order to avoid race condition  and locks during multi threaded calculation. The portions of sub blocks that are not shared between threads are calculated by different threads.

## Solutions attempted

- Tracing Scheduling

We attempted to implement more aggressive scheduling at if (!cell.isAlive) branch using tracing scheduling. We assume the branch condition is always true and only revert the changes if the assumption is proven wrong. The implementation can be summarized as the following:

```
int alive =cell.isAlive
// Do some work
if (alive)
// Repair Algorithm
else
// Done
```

However, this approach failed to demonstrate any performance improvement.

- Multithread on map_initialization

The extremely complicated dependency relationships resulted in bugs we were not able to resolve