CSA – Coursework 2021

Conway’s Game Of Life

* Student A – Rohak Roy()
* Student B – Corneci Stefan( xl20294 )

*STAGE 1 – PARALLEL IMPLEMENTATION*

1.Functionality and Design

1.1.Functionality Implemented

As requested, at first we implemented a serial, single threaded-code for the Game of Life. It involved transforming the input, a binary image, into a 2D slice, representing the world in the first stage. Using the functions implemented in the first lab, we applied the logic of the game on the slice and stored the final state of the world in a new 2D slice variable(updatedWorld).

Passing step 1, we started implementing a parallel version of the program. In the distributor function we shared the tasks to different worker threads to operate on different slices of the image(the same as the number of threads in gol.Params.Threads), in parallel. In the loop that iterates through the turns we have goroutines, each for one slice, in matters of the number of threads. Then, everything is appended to an updated world.

With the use of a ticker and the calculateAliveCells function, we could write an implementation that reports the number of cells that are still alive each 2 seconds into the events channel(using the AliveCellsCount struct).

In the last step, to visualise the state of the game we used CellFlipped and TurnComplete event. In case of s is pressed, a current state of the board is generated in a pgm file, using the ioOutput channel. The same for “q” key but the program finishes execution after generating the current image, using ImageOutputComplete and FinalTurnComplete. For “p”, we used another channel which stores a Boolean value to make sure that we can resume the game at another press of the key.

1.2. Problems solved

We had to be careful of race conditions that could appear when accessing some data. We used mutex locks for the AliveCellsCount struct that used the function to calculate the alive cells needed. Two more mutexes are implemented for the world and the TurnComplete event.

Another problem that we came across was the implementation of the control rules, especially implementing the logic of the pause button. To make it work, we made use of a channel containing a Boolean value that is reverted when the ‘p’ key is pressed again, so the execution may continue.

Therefore, our program splitting the image into slices and applying the game logic on each, concurrently, and using channels for input and output is an efficient parallel implementation of the Game of Life.

2.Testing and Critical Analysis

That is how our parallel implementation works for a 512x512 image. Benchmarking was done using the given function implemented in a new file on a 6 core- 12 threads machine.

*STAGE 2 – DISTRIBUTED IMPLEMENTATION*

1.Functionality and Design

1.1.Functionality Implemented

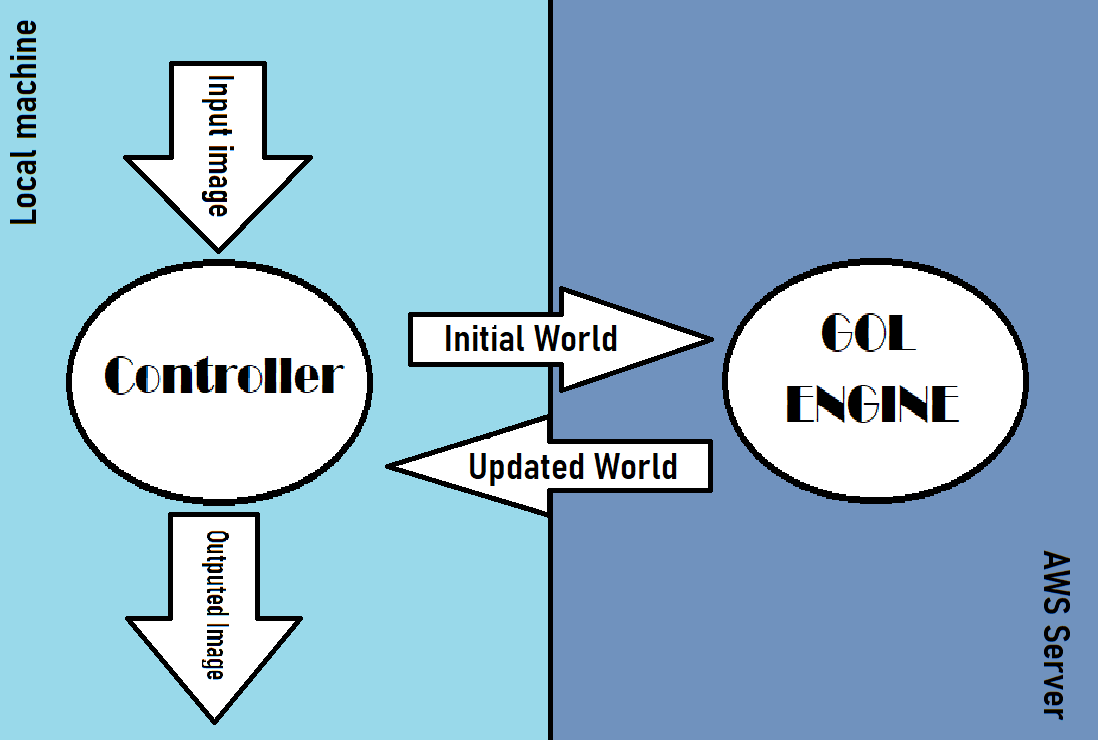
First step of our distributed implementation was splitting the

initial Code into two components: a controller and a server(AWS node). This involved setting up RPC connections between these components, using also the net package, so the state of the Game of Life are communicated between machines over a network. We created a file stubs.go which has two structs for message passing, a Request and a Response.

Inside the server file there is the functionality of Game of Life( calculateAliveCells function and calculateNextState) and we also implemented the ProcessTurns function, which processes the turns of the game. In the controller there are the makeCall functions that take a request and return a response from the server.

1.2.Problems Solved

The distributed implementation was more challenging than the parallel one and we came across a few issues. We have struggled a bit to connect the controller to the AWS node and to manipulate data through the network. Moreover, the use of mutexes for the world and for alive cells was crucial for avoiding race conditions.



2.Testing and critical analysis