**IMPROVISATION OF CONWAY'S GAME OF LIFE**

by

**Kaustubh Jha 18bce1043**

**Taksali Anjali 18bce1254**

**Vivek Gorania 18bce1310**

A project report submitted to

**Dr. Balasundaram A**

**SCOPE**

In fulfilment of the requirements for the course of

**CSE4001 – PARALLEL AND DISTRIBUTED COMPUTING**

In

**B.Tech Computer Science and Engineering**

****

**Vandalur – Kelambakkam Road**

**Chennai – 600127**

**OCT 2020**

**BONAFIDE CERTIFICATE**

Certified that this project report entitled “Improvisation of Conway's Game Of life**”** is a bonafide work of **Kaustubh Jha – 18BCE1043, Taksali Anjali – 18BCE1254** and **Vivek Gorania – 18BCE1310** who carried out the Project work under my supervision and guidance for **CSE4001 – Parallel and Distributed Computing.**

**Dr. Balasundaram A.**

Assistant Professor (Sr.)

SCOPE

VIT University, Chennai

Chennai – 600 127.

**ABSTRACT**

Conway’s Game of Life isn't simply a game, but more of an algorithm used in Cellular automation and many processors. There is nothing like winning or losing or planning in this game. Life is simply sort of a “cellular automaton” - a system of cells that survive a grid, wherever they live, die, and evolve per the principles that govern their world.

Life’s easy, elegant rules produce an amazing game of life, advanced emerging behaviour and logic behind this game. It is played on a 2-D canvas n\*m matrix where at first values are given at random then based on its rules it executes its algorithm. Each square in the grid contains a cell, and each cell starts its state in this game as either Alive or Dead and proceeds as per the rules of the game. During each iteration in the game, each cell counts its Alive neighbouring cell.

The main goal of our project is to implement Conway's game of life on a grid of n\*m cells where n and m are the number of rows and columns. This would be full VGA based, and updated as fast as the monitor can display the grid .Our design uses massive parallelization to achieve the required speeds. Also, we tried to make it user-interactive to show the game much more interesting and easier to understand the reason behind this algorithm.

**Keywords:**

Conway’s, VGA, Life, Automation, Alive

**ACKNOWLEDGEMENT**

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. Balasundaram A.,** Assistant Professor (Sr.), SCOPE, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

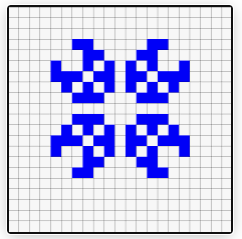
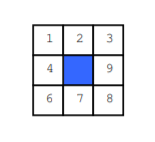
**Kaustubh Jha Taksali Anjali Vivek Gorania**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **SL. NO.** | **TITLE** | **PAGE NO.** |
| 1. | ABSTRACT | 3 |
| 2. | ACKNOWLEDGEMENT | 4 |
| 3. | INTRODUCTION | 6 |
| 4. | RELATED WORK/ LITERATURE  REVIEW | 8 |
| 5. | PROPOSED WORK | 11 |
| 6. | EXPERIMENTAL SETUP (HARDWARE & SOFTWARE SPECIFICATIONS ) | 19 |
| 7. | EXPERIMENTAL RESULT DISCUSSION | 20 |
| 8. | CONCLUSION AND FUTURE WORK | 21 |
|  | REFERENCES |  |
|  | APPENDIX 1 (Code Files as attachments) |  |
|  | APPENDIX 2 (Execution steps and screenshot of outputs ) |  |

**INTRODUCTION**

In Conway’s Game of Life which is represented in the form of n\*m matrix where n is number of rows and m is number of columns, where colors are used to show whether cells are alive or dead .Each cells on the board which is represented by a square here has 8-8 neighbours as illustrated below:



To determine the next state of each cell of the board on the next generation of the game, each cell is examined at the same time. A cell on the board continues to live if it has either two or three neighbouring cells which are in Alive state. A Cell brought in alive state if it has exactly three neighbouring cells which are in Alive state. In all the remaining state cells are either dead or remain in its Alive state. Generations continue with each iteration which may not be seen sometimes on the board, repeating the same steps till each on the board reaches its stable state either Alive or Dead and no more changes can be done as per the rules of the game.

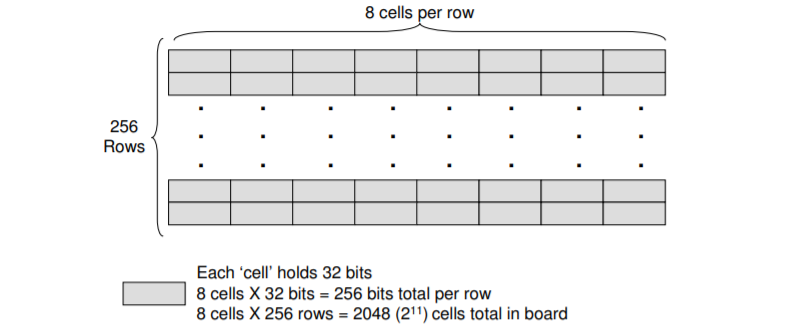
**Cells and Board Representation**

To show every component of our design required in our project, to show proper parallelization and speed of our algorithm we need a large matrix, the size of our matrix is 256x256 cells where each cells represents each pixel ,pixel size and color is scalable which can be changed .At initial state values of each are given in random deciding whether they are Dead or Alive.

**Matrix Representation in System**

Here each cell represents one bit in RAM. Cells in RAM hold 32 bits. Since it is of 256 bits in length and height, we can consider this as having 8 cells of thirty two bits per row and 256 rows in total.

The following diagram will show the bit representation in RAM.



**Parallelization**

We split the grid into 8 columns and 256 rows for testing. We picked 8 for the testing because it uses the M4K blocks efficiently, having one per column, and they can be easily split into a column reference and location. Each column has the method to do the calculation to find the next generation state and a state machine that may calculate a given row once triggered. We even have a bigger management module that controls and synchronizes these columns.

The management module is that the sole module the columns directly move with. The management module triggers all of the columns to calculate a row in parallel the instant the VGA module finishes rendering a row. This keeps the calculation and VGA in adjustment.

The management Module permits the speed of calculation to be controlled. There are heaps of your times wherever the columns don't seem to be doing any calculation that the system itself is capable of a lot of quicker speeds if we tend to were to decouple the calculation from the VGA scanning.

Here is an associated animation of however the grid is updated in parallel. As you'll be able to see, all columns end up changing the row with lots of time to spare.

**RELATED WORK/ LITERATURE REVIEW**

**a. Conway’s game of life is a near-critical metastable state in the multiverse of cellular automata**

The cellular automaton Game of LIFE had been extensively studied within the Nineties by applied math physicists. Bak, bird genus and Creutz claimed that LIFE may be a system presenting self-organized criticality (SOC) with a non-preserved amount whereas Bennett and Bourzutschky argued that the ascertained criticality was thanks to finite size effects. Since Other nonconservative SOC models have additionally had their strict essential behavior contested; many studies examined LIFE thoroughly, with the final conclusion that the game of life is slightly subcritical . Moreover, singame of life-site mean-field approximations were developed for deducing game of life densities, however nobody may reproduce the numerical results from simulations in the square. lattice. Therefore, it absolutely was believed that mean-field approximations were not applicable to LIFE and weren't terribly helpful to cellular automata (CA) rules generally.One decade before, metallic elements projected a qualitative classification for CA behavior. This classification consists of the category I (fixed point), category II (periodic), ClassIII (chaotic) and sophistication IV (“complex”) behaviors. However, Wolfram’s categories square measure solely phenomenological descriptions: given a CA rule, it's impossible to predict to that category it pertains to. An effort during this prophetic direction was created by Langton , who proposed the parameter λ to classify the CA rules, which, sadly, failed at describing complex rules like LIFE.

In this context, the queries that we wish to explore square measure the subsequent. In what sense is the game of life essential (or subcritical)? is the singame of life-site mean-field approximation not applicable to LIFE and alternative “complex” rules? Is there any parameter for CA rule house (similar to a control parameter and obtained a priori from the rule table) to order the CA rules and reveal any physical change? What quite a transition is that the game of life connected to? Our principal findings concern the quality of the singame of life-site mean-field (MF) approximation. We discover that this type of medium frequency approximation is often applied to clarify the density of live cells within the game of life and to outline a brand new management parameter. we have a tendency to describe LIFE behavior in terms of existence and competition between 2 phases and show that it corresponds

to a subcritical (but quasi critical) nucleation method of living cells. For an outsized variety of CA, we have a tendency to find that the medium frequency predictions square measure qualitatively and even quantitatively correct. Within the topological space of the 6144 order three rules (rules that have the medium frequency equation dominated by ρ 3 once 0), the medium frequency analysis used here predicts that 22032 of them have solely a trivial zero part. For the remaining 3941, the medium frequency predicts a nontrivial phase ρ∗ which can not be stable to invasion by the zero part once simulated in an exceedingame of lifey sq. (2D) lattice. These 3941 rules include 440 automata that have amount T ≥ two and are excluded from our study. within the remaining 3501 rules, most of the CA patterns found in an exceedingame of lifey square lattice are often viewed as composed of vacuum and alive part domains.

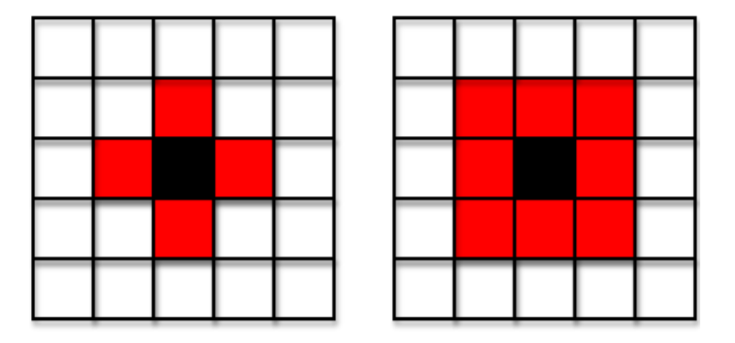
**b. Parallelization: Conway’s Game of Life By Aaron Weeden, Shodor Education Foundation, Inc.**

The cellular automaton is an important tool in science that can be used to model a variety of natural phenomena. Cellular automata can be used to simulate brain tumor growth by generating a 3-dimensional map of the brain and advancing cellular growth over time . In ecology, cellular automata can be used to model the interactions of species competing for environmental resources. These are just two examples of the many applications of cellular automata in many fields of science, including biology , ecology , cognitive science , hydrodynamics , dermatology, chemistry ,environmental science , agriculture , operational research, and many others.

Cellular automata are so named because they perform functions automatically on a grid of individual units called cells. One of the most significant and important examples of The cellular automaton is John Conway’s Game of Life, which first appeared in . Conway wanted to design his automaton such that emergent behavior would occur, in which patterns that are created initially grow and evolve into other, usually unexpected, patterns.

He also wanted to ensure that individual patterns within the automaton could dissipate, stabilize, or oscillate. Conway’s automaton is capable of producing patterns that can move across the grid (gliders or spaceships), oscillate in place (flip-flops), stand motionless on the grid (still lifes), and generate other patterns (guns). Conway established four simple rules that describe the behavior of cells in the grid. At each time step, every cell in the grid has one of two particular states: ALIVE or DEAD.

The rules of the automaton govern what the state of a cell will be in the next time step. Like all cellular automata, the rules in Conway’s Game of Life pertain to cells and their “neighbors”, or the cells to which a cell is somehow related (usually spatially). The collection of a cell’s neighbors is known as the cell’s “neighborhood”. The code in this module uses the latter of these two, the “Moore neighborhood”, in which the 8 cells immediately adjacent to a cell constitute the cell’s neighbors.



**Two types of neighborhoods of a black cell, consisting of red neighbor cells: von Neumann neighborhood (left) and Moore neighborhood (right).**

The automaton begins by initializing the states of the cells every which way and ends when a certain variety of your time steps have passed on. For a ttiny low board, a cellular automaton will be simulated with a pencil and paper. For even a fairly little board, however, operating by hand becomes cumbersome and also the use of a pc is required to run the simulation in an appropriate quantity of your time. A coded implementation of Conway’s Game of Life running on one pc will simulate a fairly massive grid in a {very} very bit of your time. To simulate an excellent larger grid, one has to use a lot of process power than is available on one processor. The idea of data processing will be wont to leverage the machine power of computing architectures with multiple or several processors operating along. Conway’s Game of Life is a remarkable drawback to lay attributable to the boundary conditions involving ghost rows and columns and since neighbor cells square measure tightly-coupled; that's, neighbors relate to every alternative directly and need calculations to be performed on teams of them. This ends up in fascinating parallel communication patterns and needs a modest quantity of computation similarly.

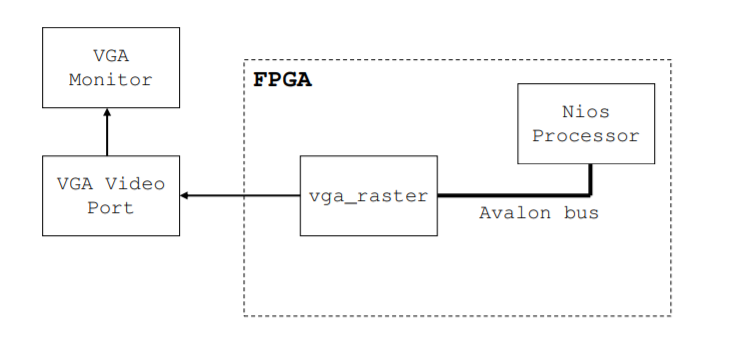
**PROPOSED WORK**

In this project we have shown implementation of Conway’s Game of Life which has Html and Css as the frontend and P5 framework of javascript as backend.

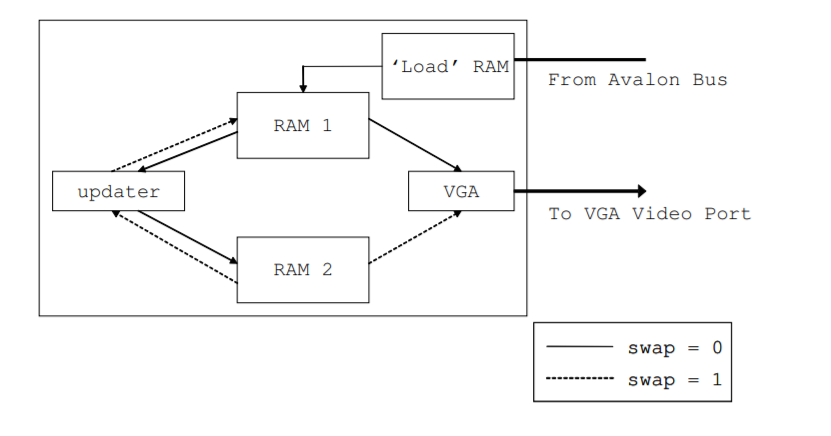
The initial goal of our project is to create a function putting random values in the grid. This function will access 2 input sets previously while setting canvas resolution that is width and height. It will return the current state of each cell representing each pixel in the board or matrix a board state in which every cell has been randomly initialized to either ALIVE or DEAD. We can choose any color to represent each state, here we have taken white and black to show better animation. In Life these random patterns are known as “Soups”, and they are the quickest way for us to start producing interesting output.

The Life world is a 2-D grid. Regardless of language, 2-D grids are almost always represented in code using a list of lists

**This is the system diagram of our project**:

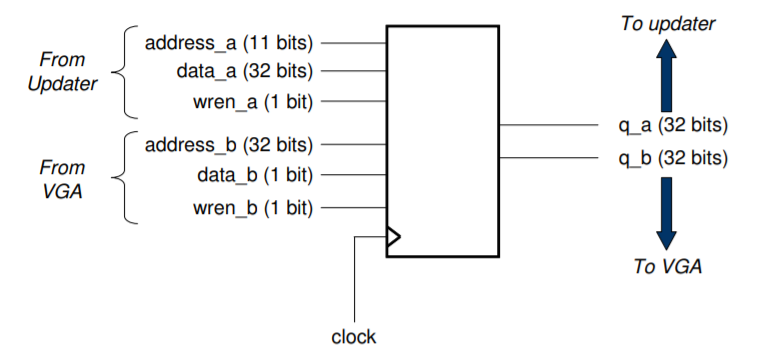


The Architecture of our project is very simple. The Nios Processor is responsible for sending the user initiated conditions to the vga\_raster through the Avalon bus here the condition signifies the rules we are setting.



**Figure. Detailed look of vga\_register**

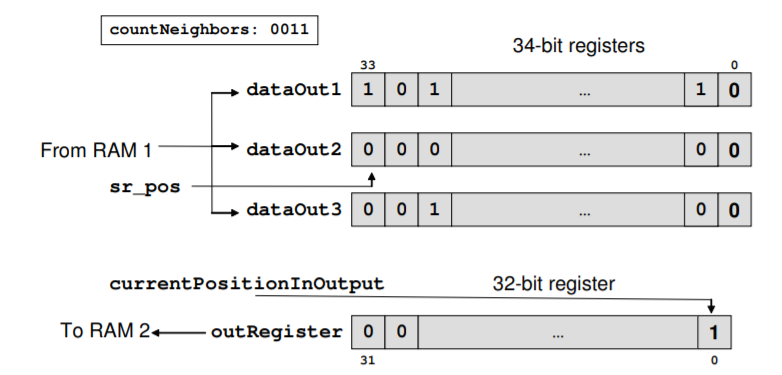
The vga\_raster in this architecture serves three functions. The first as shown is accepting initial conditions from the Nios Processor by the Avalon Bus. The second is, it’s taking the current state of each cell in the board and displaying it on screen through the ‘VGA’ block. The third but very important, is to update the current state of the cell and display the next state of the cell to another block of RAM.



**Figure. Design contain two RAM component**

**Game Logic**

The main function in this game is done by the updater component of Vga\_register. The following diagram will show how the system updates this board game from current state to next state.



There are three 34-bit shift registers that are meant to carry the contents of 1 RAM one cell (as mentioned earlier on). The two bits of excess are needed for the following time the register is loaded with the adjacent cell’s bits. The rationale we tend to load 3 rows is as a result of we want to look at the eight close neighbors of every cell in order to work out its life or death for the following generation. The variable countNeighbors holds the worth of what number close neighbors are alive (hold a worth of 1). betting on whether or not this organism we tend to be examining is dead or alive, the acceptable next generation price is written to a 32-bit register. Upon filling the 32-bit register with information, this register is then written to RAM two.

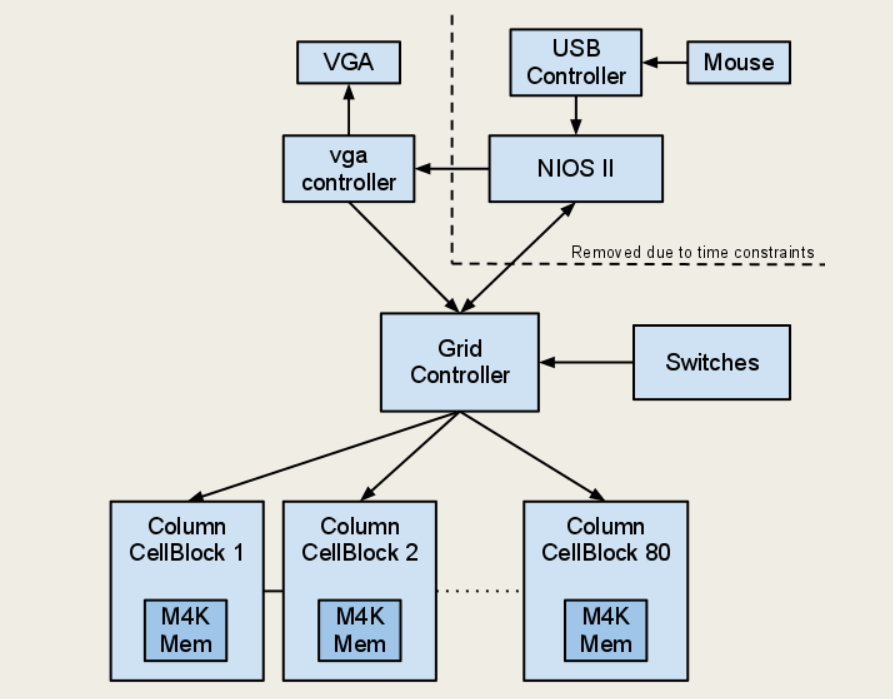


Figure.Detailed Diagram

**Parallelization**

In order to use parallelism, the hardware component should support it. The classical computer first established by Von Neumann has only a single Cpu connected to memory. As if such kind of architecture does not support parallelism because there will be only one Cpu to run all instructions .

It ought to be noted that shared memory is admittedly simply a style of quick message passing. Threads should communicate, even as processes should, however threads get to speak at bus speeds (using the front-side bus that connects the C.P.U. to memory), whereas processes should communicate at network speeds (ethernet, InfiniBand, etc.), that square measures a lot slower.

Threads may also have their own non-public recollections, and OpenMP has constructs to outline whether or not variables square measure public or non-public to threads.

There are unit compelling benefits for exploitation similarity. The 3 motivations considered during this module area unit acceleration, accuracy, and scaling.

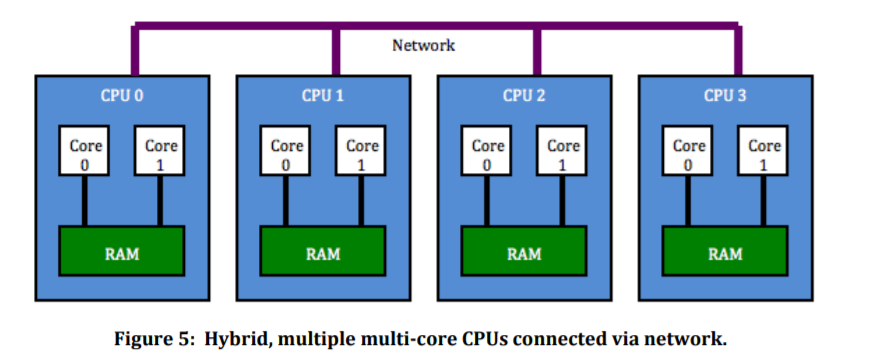
“Speedup” is something that shows how fast a program can be executed which could be done easily with the help of parallelism.

“Accuracy” is something that shows how close we are with the required goal.

Last is “Scaling”. Scaling is that the concept that additional parallel processors may be wont to model a much bigger downside within the same amount of your time it might take fewer parallel processors to model a smaller downside.

**Memory:**

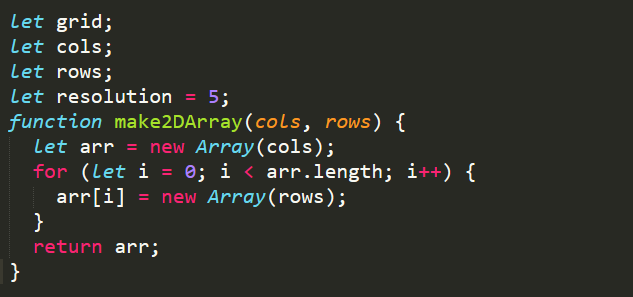
Due to the speed necessities, we took M4K blocks, as they're twin ported (both the VGA and my Hardware will read/write) and quick. they will even be created extremely parallel exploitation one block per parallel unit.



**There are basically 5 Steps of our project:**

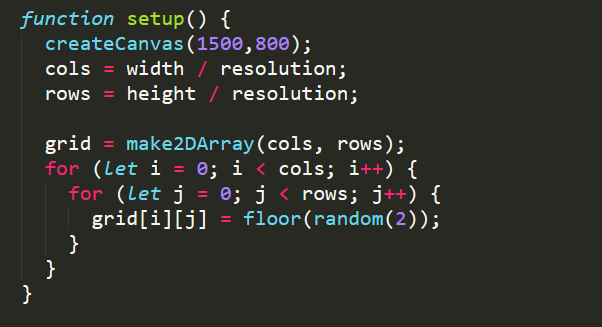
1. **Creating our board**

Taking input as number of Rows and Columns

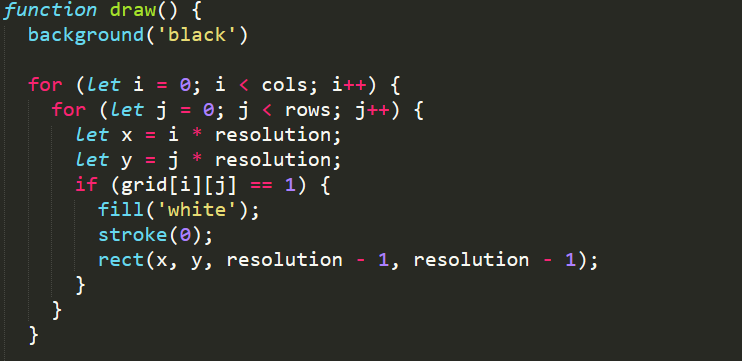


1. **For setting size of each cell and it’s random value(dead or Alive)**

Scaling of size and resolution can be done here



1. **For background and cell effect**



1. **Parallel processing for finding next state of all cells at same time**



**Rules to Play Game:**

The rules of the game are simple, and describe the evolution of the grid:

◮ Birth: a cell that is dead at time t will be alive at time t + 1 if exactly 3 of its eight neighbors were alive at time t.

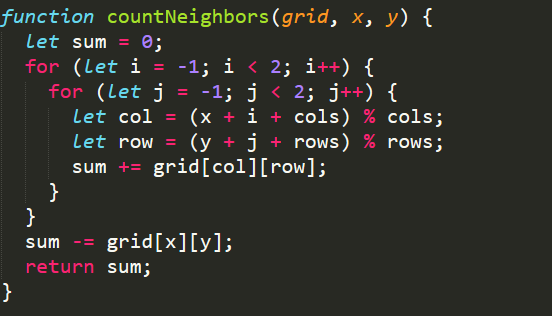
◮ Death: a cell can die by:

◮ Overcrowding: if a cell is alive at time t + 1 and 4 or more of its neighbors are also alive at time t, the cell will be dead at time t + 1.

◮ Exposure: If a live cell at time t has only 1 live neighbor or no live neighbors, it will be dead at time t + 1.

◮ Survival: a cell survives from time t to time t + 1 if and only if 2 or 3 of its neighbors are alive at time t.

1. **Returning number of live neighbours**



**EXPERIMENTAL SETUP**

**(HARDWARE & SOFTWARE SPECIFICATIONS)**

**Software Setup:**

**FrontEnd:**

1. Html

2. Css

3. JavaScript

**Backend:**

1. P5 framework of JavaScript

2. OpenMp

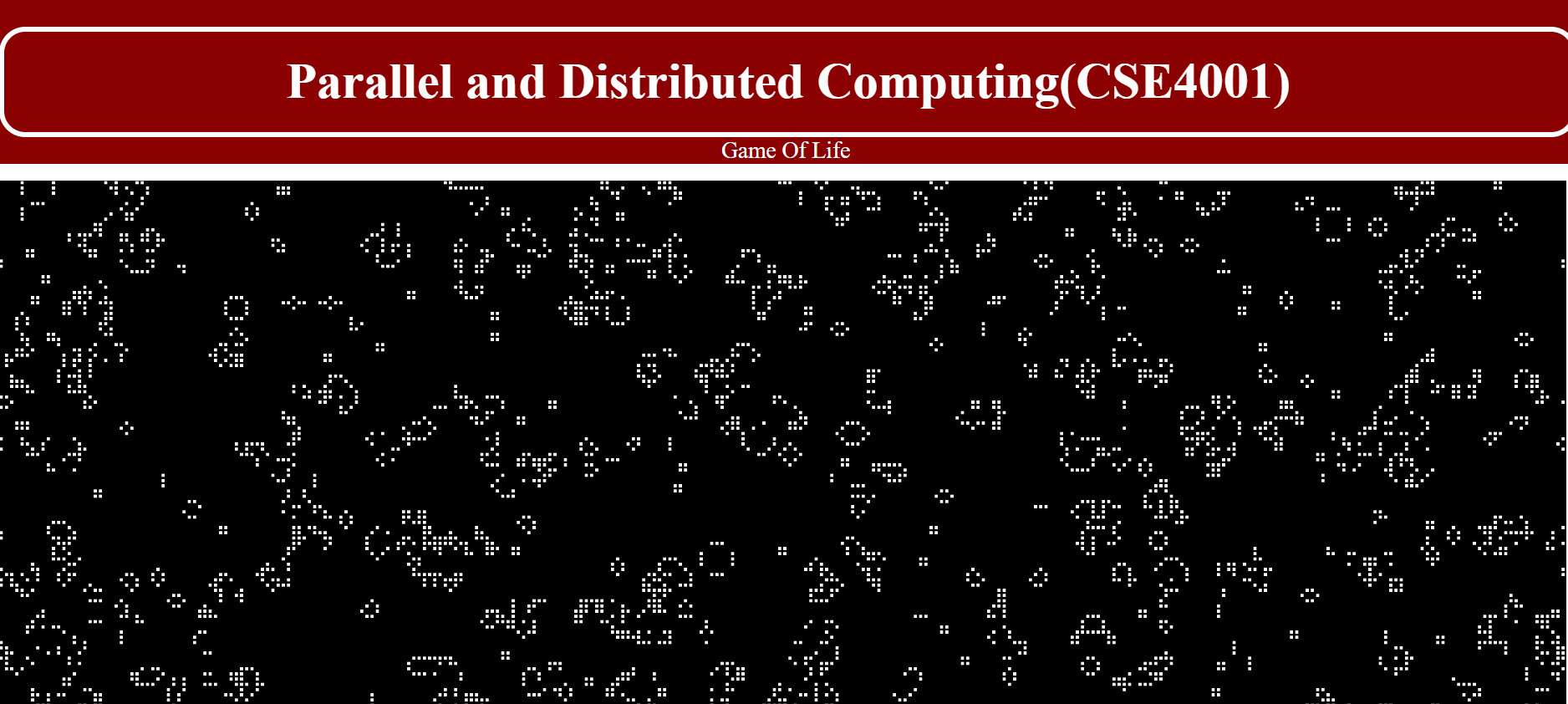
**Hardware**

**VGA**

The main work of VGA component is to properly display bits from RAM on screen. Since Our matrix size is 256\*256 where each pixel represents a pixel, to determine whether a cell will Live or Die we have to read each bit content of the RAM. If bit obtained is 1 we color it white else black.

**EXPERIMENTAL RESULT AND DISCUSSION**

**Initial State:**

****

In this project we have successfully implemented Conway’s game of Life Algorithm. Our Design and execution of the algorithm went through many ideas and changes and finally came up with an improvised and quite simple approach to Conway’s game of life algorithm.

The key learning for us was, we start focusing more on clock cycles and memory because timing analyzer is one of the most considered criteria in this algorithm.

An issue arises while dealing with cells present on the edges of the grid since they do not have full 8 neighbouring cells, they have only 5 and those present in corners have only 3, in our project we resolved all such issues.

**CONCLUSION AND FUTURE WORK**

As demonstrated in Appendix I and II, we have successfully parallelized Conway's Game of life. The speedup achieved will significantly improve the performance of Cellular automata based applications. We used the P5 framework in JavaScript to visually demonstrate its working. Therefore, in case of future works, there is a possibility of using other platforms to improve the speedup achieved.

**Subject Focused on:**

* Scale parallel code
* Cellular Automata
* Parallel hardware design
* Html
* OpenMP
* P5 framework

**APPENDIX-I**

**HTML, CSS CODE:**

<!DOCTYPE html>

<html>

<head>

<style>

div.a{

width:97%;

border-radius: 25px;

border: 5px solid #ffffff;

background-color:#8b0000;

padding: 20px;

font-size:48px;

text-align: center;

}

div.c{

font-size:22px;

width:100%;

height:18%;

background-color:#8b0000;

text-align: center;

}

</style>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Game of Life</title>

</head>

<body>

<div class="c"><br><div class="a"><b><font face='open sans' color=#ffffff>

Parallel and Distributed Computing(CSE4001)

</font></b></div>

<font face='open sans' color=#ffffff>

Game Of Life

</font>

</div><p>

<div style="width:800px; margin:0 auto">

<script type="text/javascript" src="https://cdn.jsdelivr.net/npm/p5@1.1.4/lib/p5.min.js"></script>

<script align="content" type="text/javascript" src="sketch.js"></script>

</div>

</body>

</html>

**JAVASCRIPT CODE:**

let grid;

let cols;

let rows;

let resolution = 5;

function make2DArray(cols, rows) {

let arr = new Array(cols);

for (let i = 0; i < arr.length; i++) {

arr[i] = new Array(rows);

}

return arr;

}

function setup() {

createCanvas(1500,800);

cols = width / resolution;

rows = height / resolution;

grid = make2DArray(cols, rows);

for (let i = 0; i < cols; i++) {

for (let j = 0; j < rows; j++) {

grid[i][j] = floor(random(2));

}

}

}

function draw() {

background('black')

for (let i = 0; i < cols; i++) {

for (let j = 0; j < rows; j++) {

let x = i \* resolution;

let y = j \* resolution;

if (grid[i][j] == 1) {

fill('white');

stroke(0);

rect(x, y, resolution - 1, resolution - 1);

}

}

}

let next = make2DArray(cols, rows);

// Compute next based on grid

for (let i = 0; i < cols; i++) {

for (let j = 0; j < rows; j++) {

let state = grid[i][j];

// Count live neighbors!

let sum = 0;

let neighbors = countNeighbors(grid, i, j);

if (state == 0 && neighbors == 3) {

next[i][j] = 1;

} else if (state == 1 && (neighbors < 2 || neighbors > 3)) {

next[i][j] = 0;

} else {

next[i][j] = state;

}

}

}

grid = next;

}

function countNeighbors(grid, x, y) {

let sum = 0;

for (let i = -1; i < 2; i++) {

for (let j = -1; j < 2; j++) {

let col = (x + i + cols) % cols;

let row = (y + j + rows) % rows;

sum += grid[col][row];

}

}

sum -= grid[x][y];

return sum;

}

**OPENMP CODE:**

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#include<omp.h>

#define N 30

int grid[N][N],next[N][N];

int check\_cell(int row,int col)

{

int live=0;

if(row-1>=0||row+1<N||col-1>=0||col+1<N)

{

live=grid[row-1][col]+grid[row+1][col]+grid[row][col-1]+grid[row][col+1]+grid[row-1][col-1]+grid

[row-1][col+1]+grid[row+1][col+1]+grid[row+1][col-1];

}

if(live<2||live>3)

return 0;

if(grid[row][col]==1 &&(live==2||live==3))

return 1;

if(grid[row][col]==0 && live==3)

return 1;

return 0;

}

void cpy()

{

int i,k;

for(i=0;i<N;i++)

{

for(k=0;k<N;k++)

grid[i][k]=next[i][k];

}

}

int main()

{

int i,k;

for(i=0;i<N;i++)

{

for(k=0;k<N;k++)

grid[i][k]=rand()%2;

}

int lifetime=10;

printf("Input\n");

for(i=0;i<N;i++)

{

for(k=0;k<N;k++)

printf("%d ",grid[i][k]);

printf("\n");

}

for(i=0;i<lifetime;i++)

{

#pragma omp parallel num\_threads(4)

{

int cur\_thread=omp\_get\_thread\_num();

int ipt=N/4;

int start=cur\_thread\*ipt,end=0;

end=(cur\_thread==4-1)?N:start+ipt;

for(i=start;i<end;i++)

{

for(k=0;k<N;k++)

next[i][k]=check\_cell(i,k);

}

}

cpy();

}

printf("Output\n");

for(i=0;i<N;i++)

{

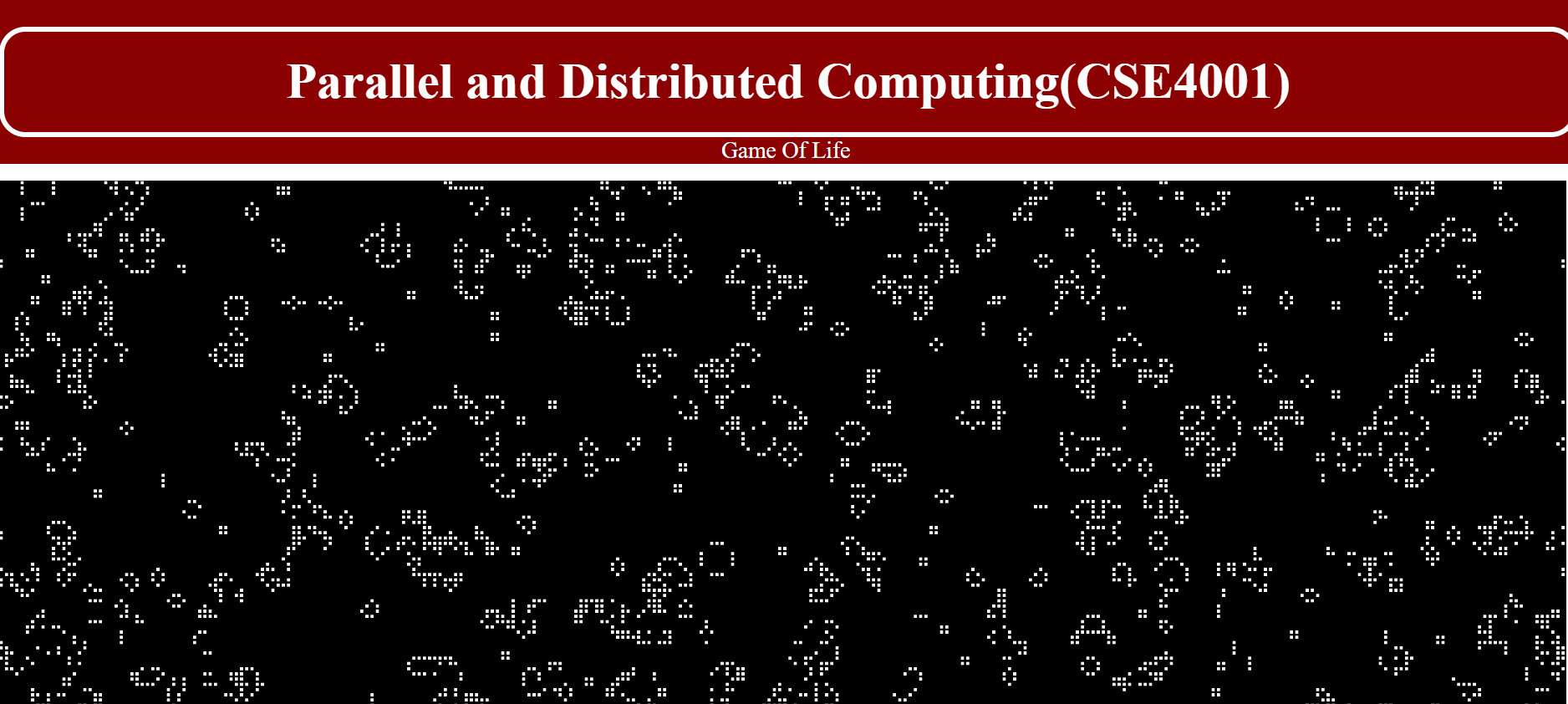
for(k=0;k<N;k++)

printf("%d ",next[i][k]);

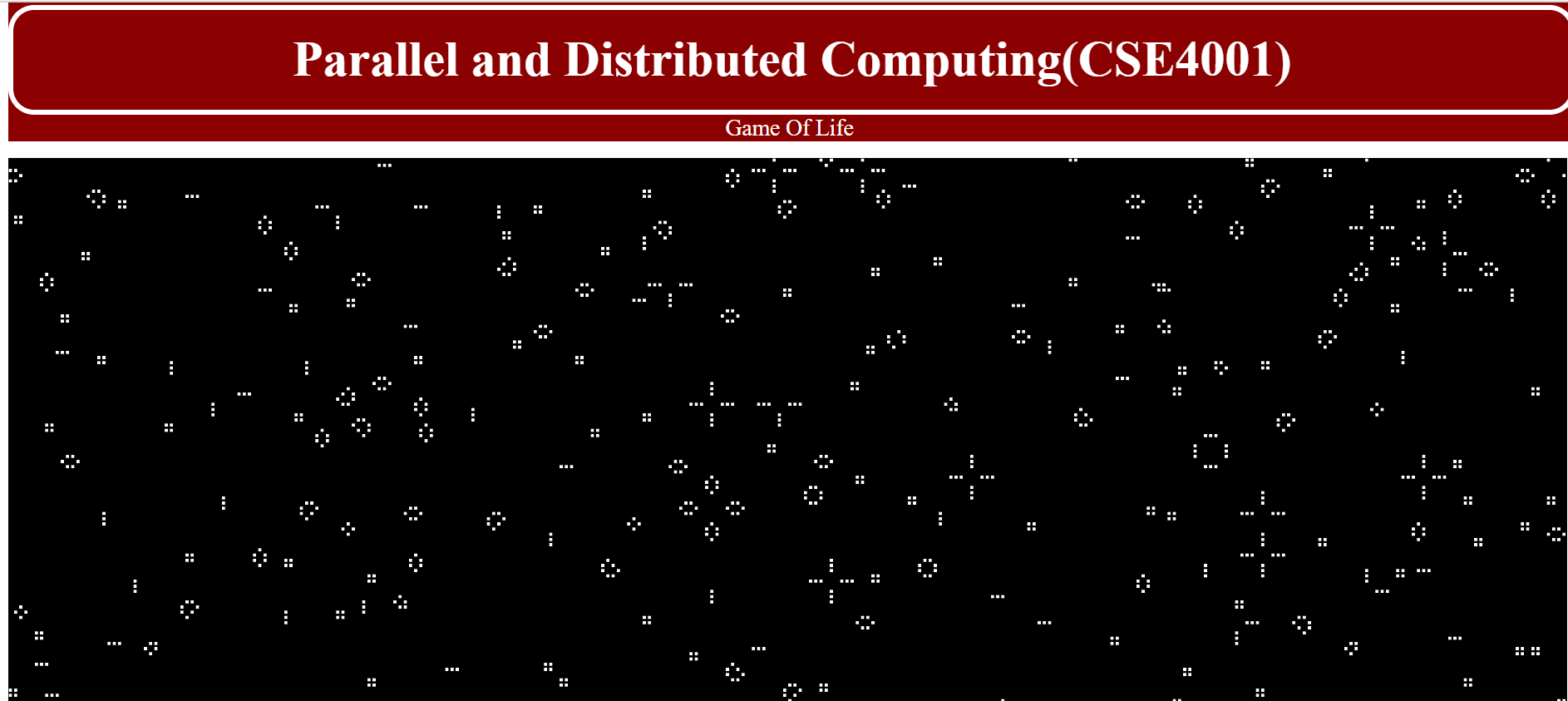
printf("\n");

**APPENDIX-II**

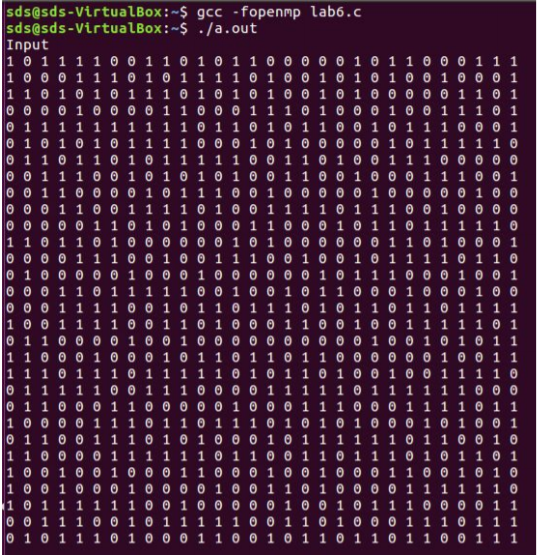
**Initial state:**

****

**Final and stable state:**

****

**OpenMp output:**

****

****

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***