Life Game Design in CUDA

Running existing Cellular Automata in parallel environments

Zheng Xia
Electrical and Computer Engineering
University of Ottawa
Ottawa, Canada
zxia010@uottawa.ca

***Abstract*—A novel approach is designed to improve the performance of Conway life game, which is an existing cellular automata model has been defined in this paper. Meanwhile the develop Cell-DEVS tools CD++ is used as a reference during the experiment. Specifically, the three-fold experiment demonstrates the model being executed in different amount of processors and presents some convincing results based on the individual rules. In addition, another type of CA model such as modified von Neumann CA model is also test in the new platform with running in parallel. Overall, the calculate speed have been boosted up as twice as the original speed. That is, the time consuming in the experiment have a dramatic reduce near one half of the usual calculation time.**

Keywords—Cellular Automata; Parallel computing; CUDA; Conway life game

I. INTRODUCTION

Over the decades, Cellular Automata (known as CA) has become increasingly present in solving general-purpose problems yet most of the focuses are on the mapping problem. The study of cellular automata was started by von Neumann and Stanislaw Ulam in the middle of the 20th century [1]. Generally, the automata includes some cells on a two dimension grid and the states of these cells. The states are determined by a transition function that computes from states of the cell's neighbor in previous time span. Following von Newman opinions, Ulam[1] who devised a two-dimensional imaginary finite lattice for one-cellular machine formed of components named cells raised some propositions

The cellular automata model such as life game usually run in a serial based procedure, however, when the scale of the problem is increasing, the computation time on the problem will become dramatically large if the model is still running in a single processor. Due to the locality and uniformity of the cellular automata, one can adjust the transition function to fit the multi-processors platform such as CUDA and OpenMP.

In this project, the serial based processing on the cellular automata would be demonstrated firstly as a reference in the transformation. Then, the model would be tested by using the Conway life game rules. After the method is being proved in stability, a parallel life game model would be built and implemented on CUDA platform to speed up the calculation process for each cell. The results demonstrate an enhancement on the calculation process and a highly convincing mappings strategy from the CD++ to the Visual Studio and CUDA platform.

The report would be organized as follows. In Section 2, the background of the cellular automata would be discussed as well as the parallel programming platform that have been used in the experiments. A formal specification of Conway life game Cellular Automata model would be discussed in Section 3. Section 4 shows the results of the experiment using the different initial values in the model and the result of testing the modified von Neumann Cellular Automata model.

II. BACKGROUND

A. Cellular Automata (CA)

A cellular automaton (as known as CA) is a discrete model that has been studied through various domains such as computability theory, mathematics, complexity science and etc. [1] Specifically, a cellular automaton is made by cells in a typically lattice representing the finite states of the cell. In addition, the grid can be in any existing dimensions. The neighbor of the cell, which plays a vital role in the automaton, decide the direction of evolution in the game. That is, the state of the neighbor and the cell decides the state of the cell itself according to some presetting rules. Generally, the rules for the updating is similar to the others and it usually does not change too much during the iteration. The main purpose for inventing the automata was to solve Self-Replication problem in a system.

Besides, the cellular automata could be widely used in

- Computer processors

This means that the machine can process information computationally by implementing the cellular automaton processors with CA concepts physically.

- Cryptography

It also shows that the cellular automaton could be used for random number generation in 2D. Typically, Rule 30 is able to encode the context with Block cipher algorithm.

- Correcting error code

Cellular Automata could also be implemented for correcting the code. In the paper written by Chowdhury et al. [2], a new scheme of building the SEC-DED codes using the proper CA model has been demonstrated. Meanwhile, it also shows an ability on decoding the code in a fast way.

B. *Parallel programming*

In computer software, a parallel programming model is a model which are compiled and executed via different processor basically. The value of a programming model can be judged on its universality: how well a variety of different architectures can express a range of different problems, and its performance: how efficiently of the problem being executed. The implementation of a programming model can take several forms such as libraries invoked from traditional sequential languages, language extensions, or complete new execution models. [3]

CUDA, also named as Compute Unified Device Architecture, is a parallel computing platform and programming model that implemented by NVDIA company by providing the access to the virtual instruction set and the usage of the memory in graphic processing units (GPUs) [4]. By using this programming language, a serial of independent data could be calculated through various threads concurrently. This will boost up the throughput of a general program in order to maximize the efficiency of the running program.

Moreover, the classic CA model was enhanced to Global Cellular Automata (GCA) by Rolf Hoffmann et al. [1] In the new model, it is not limited to a local neighborhood, which means that they can connect across to the various domain of the automata.

III. MODELS DEFINED

In this section, a formal specification cellular model of the life game will be discussed in the first place, followed by the rule that implements on the model. Mainly, there are two different rules applied on the general model — Conway's and Von Neumann's cell rule. The difference between these two are the numbers of the neighbor. The details are given as follows.

A. *Formal specification of the Conway and modified von Neumann's neighbor Cellular Automata model*

The Cell Spaces of the CA model have been defined in Figure 3.1. Notes that the graph below illustrates the Conway's and von Neumann's neighbor of a cell (shown in grey zone and black stripe).

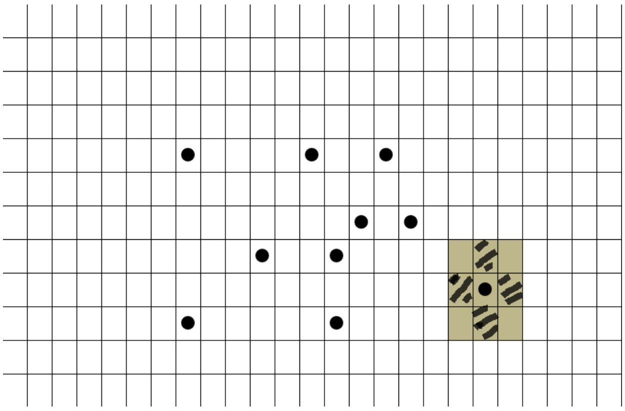


Figure 3.1 Cells and neighbour

The more concrete and specific Cellular models is defined as follows:

$$CCA = < S, n, C, \eta, N, T, \tau>$$

Where

- **C** cell's state variables;
- **S** finite alphabet to represent each cell's state;
- **η** dimensional space;
- **N** neighboring cells;
- **T** global transition function;
- **τ** local computing function;

The basic procedure for simulating a typical cellular automaton follows the discrete time simulation algorithm. The procedure flow could be defined as follows:

The steps of executing cellular automaton

1. Scan all cells
2. Apply state transition function to each cells
3. Save the next state into a different memory space
4. When all next states have been computed, the time advance, followed by constructing the next global state data.
5. Repeat the steps from 1 to 4 until the simulation ends.

Note that the dead state is sometime called quiescent state which means that if the cell and all its neighbors are in the quiescent state, its next is also quiescent. i.e. There is no change on the next step for the current cell. In this way, the discrete event simulation algorithm focuses more on processing the events rather than the cells of those with inherently more efficient.

In Conway's model, the neighbor's illustration in Figure 3.2 describes one part of the step 2. The neighborhood that is used to calculate the next states of the current cell.

N(-1, -1)	N(-1, 0)	N(-1, 1)
N(0, -1)	Cell(0, 0)	N(0, 1)
N(1, -1)	N(1, 0)	N(1, 1)

Figure 3.2: Conway's neighbor

Furthermore, for any cell, there are two possible states, either live (the black spot on Figure 3.1) or die (represented in the white color). [1][2][3] The rules of transaction, also called the decision of state machine, is depended on these four atomic action:

1. The cell is alive (i.e. Cell(0,0) = 1) and the numbers of the living neighbor $N_{i,j}$ strictly less than 2; the cell will die.
2. The cell is alive and the numbers of the living neighbor $N_{i,j}$ are 2 or 3; the cell will keep alive.
3. The cell is alive and the numbers of the living neighbor $N_{i,j}$ are more than 3; the cell will die.
4. The cell is die or does not exist and the numbers of the living neighbor $N_{i,j}$ are equal to 3; the cell would be generated (become alive).

These four rules demonstrate how the live is decided which includes rare lives condition (rule #1), normal lives condition (rule #2), overabundance lives condition (rule #3) and the propagating live condition (rule #4). In addition, the transaction of the life consist of rules above and the neighbor being used to calculate and this makes the disparity outcomes between the Conway and Von Neumann, in which Von Neumann only have 5 neighbors (Figure 3.3).

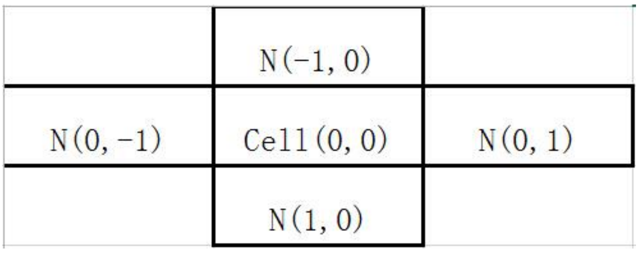
In this CA model, the original transition function can be defined as

1 The cell is alive (i.e. Cell(0,0) = 1) and the numbers of the living neighbor $N_{i,j}$ strictly less than 2 or more than 3; the cell will die.

2 The cell is alive and the numbers of the living neighbor $N_{i,j}$ are 2 or 3; the cell will keep alive.

3 The cell is dead and the numbers of the living neighbor $N_{i,j}$ are equal to 2; the cell will be re-borned.

4 The cell is dead or does not exist and the numbers of the living neighbor $N_{i,j}$ are not equal to 2; the cell would keep the states.



	N(−1, 0)	
N(0, −1)	Cell(0, 0)	N(0, 1)
	N(1, 0)	

Figure 3.3: Von Neumann's neighbor

After the formal specification of the cellular automaton has been defined, the actual processing could run on the model itself. The following section discusses how the neighbors are calculated in the real program transformation.

B. *Single processing model*

For saving the cell spaces, the 2D matrix will give the exact view as the definition in CD++. The definition of cell model in single processor is shown as follows

Cell[*i*][*j*]

Where the *column* represents the width of the 2D cell's map. The *i* and *j* is the iteration direction of lateral and vertical, respectively. Here, we could also use the 2D matrix which show the cell space distribution directly yet it does not support for the large scale problem solving when the rows and columns over 1000. That is the system limitations on the 2D array. Alternatively, one dimension array could have done the same work only if the correct configuration is given. The definition of the cell space in 1D is described as follows

Cell[*i*column+j*]

For the calculation of the neighbor states through the cellular model on boarder conditions, it is well-defined in CD++, yet in other types of platform, the general and specific approaches is decided as follows

- Top left corner

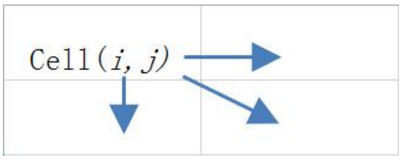


Figure 3.4a): Neighbor includes: (i,j), (i+1,j), (i,j+1),(i+1,j+1)

- Top right corner

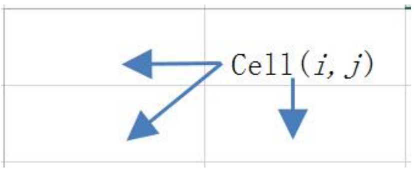


Figure 3.4b): Neighbor includes: (i,j),(i+1,j), (i,j-1),(i+1,j-1)

- Top boarder

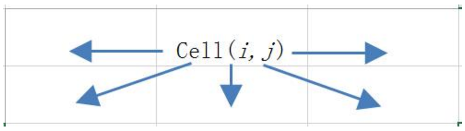


Figure 3.4c): Neighbor includes: (i,j),(i,j-1),(i,j+1), (i+1,j),(i+1,j-1),(i+1,j+1)

- Left boarder

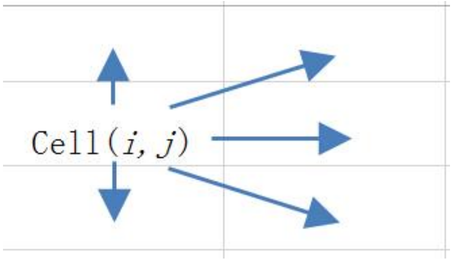


Figure 3.4d): Neighbor includes: (i,j),(i+1,j),(i-1,j), (i,j+1),(i-1,j+1),(i+1,j+1)

- Right boarder

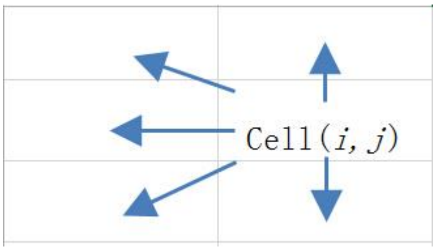


Figure 3.4e): Neighbor includes: (i,j),(i+1,j),(i-1,j), (i,j-1),(i-1,j-1),(i+1,j-1)

- Bottom boarder

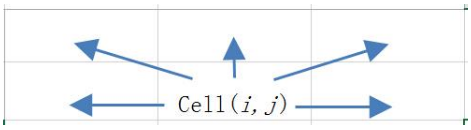


Figure 3.4f): Neighbor includes: (i,j), (i+1,j),(i-1,j), (i,j-1),(i-1,j-1),(i+1,j-1)

- Bottom left corner

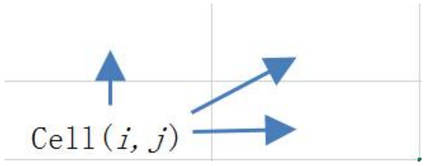


Figure 3.4g): Neighbor includes: (i,j),(i,j+1), (i-1,j),(i-1,j+1)

- Bottom right corner

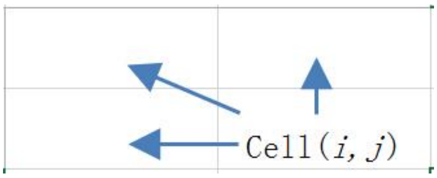


Figure 3.4h): Neighbor includes: (i,j),(i,j-1), (i-1,j),(i-1,j-1)

- General cell

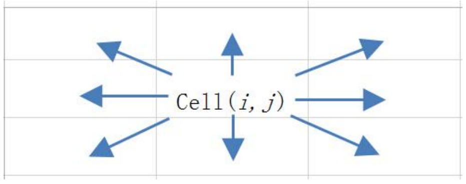


Figure 3.4i): Neighbor includes: (i,j), (i±1,j±1)

By combining the counting rules illustrated above with the one dimension array shown before, one can easily construct a different version of the Conway life game.

The counting of the von Neumann neighborhood, however, is shown below.

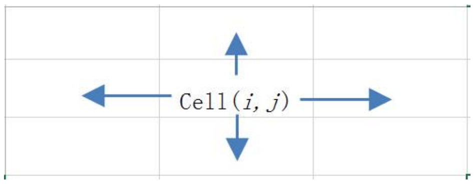


Figure 3.4j): Neighbor includes: (i,j), (i+1,j),(i-1,j),(i,j-1),(i,j+1)

It shows that the Von' neighbor does not count the diagonal cell as its neighbor.

Another problem we faced is that the judgment of the boarders. It is boards that meet the definition problem in the most of cases. Hence, for the 1D array, the judgments are show as follows

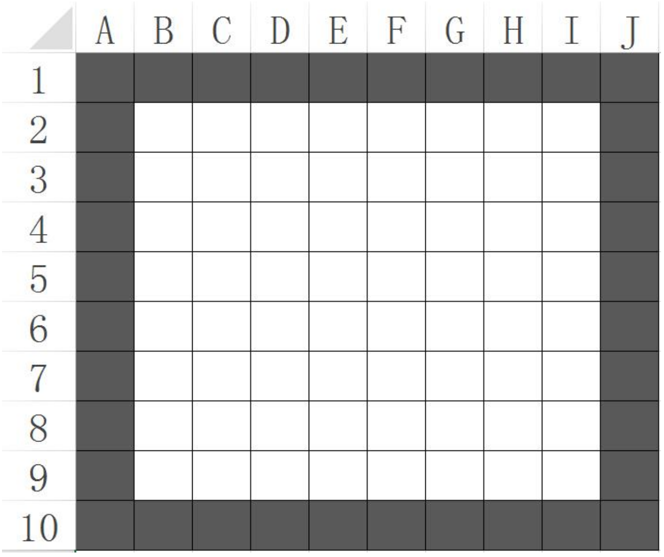


Figure 3.5: cell space boarder (with size of 10*10)

Taking the 10*10 matrix as an example shown above, the boarder's settings are:

- 1A is the top left corner
- 1J is top right corner
- 10A is the bottom left corner
- 10J is the bottom right corner
- 1B to 1I are the top boarder
- 2A to 9A are the left boarder
- 2J to 9J are the right boarder
- 10B to 10I are the bottom boarder

By using *i* and *j* represents the row and column, respectively. The condition for the boarders and corners are as follow

- 1A: *i* == 0 && *j* == 0
- 1J: *i* == 0 && *j* == (column - 1)
- 10A: *i* == (row - 1) && *j* == 0
- 10J: *i* == (row - 1) && *j* == (column - 1)

- 1B to 1I: i == 0 && j != 0 && j != (column - 1)
- 2A to 9A: j == 0 && i != 0 && i != (row - 1)
- 2J to 9J: j == (column - 1) && i != 0 && i != (row - 1)
- 10B to 10I: i == (row - 1) && j != 0 && j != (column - 1)

C. Parallel processing model

The structure of parallel model in CUDA is shown as below

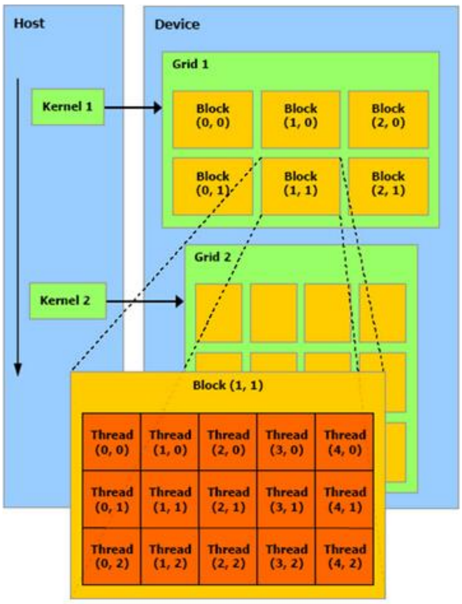


Figure 3.6: Kernel structure of running association in device

The host issues a succession of kernel invocation to the device. Each kernel is executed as a grid of thread blocks. [6] The model in this problem could be defined as follows

$$\text{Cell}[id]$$

where id = blockIdx.x * blockDim.x + threadIdx.x

Note that the each block is organized as 3D array of threads (i.e. blockDim.x, blockDim.y and blockDim.z). The *id* in this case considers the block index within the grid (blockIdx.x) and dimensions of the block (blockDim.x) and thread index within the block (threadIdx.x). Specifically, *id* represents the ID of the thread that have been designated to calculate the state counts for a typical cell itself. Because of the uniqueness of thread ID, this could be realized to process the calculation regardless of the environment around the target. Therefore, this makes the core of the parallel computing in CUDA platform. In addition, same calculation methods have been implemented in the kernel function (show in below) which are the same as the Figure 3.4a to 3.4j including the modified von Neumann neighborhood calculation shown in previous section [5][11]. And the execution function is

__global__ lifeGame<<<*Dg*,*Db*,*Ns*>>>(arguments)

where *Dg* represents the dimension and size of the grid; *Db* is the dimension and size of each block; *Ns* is an optional setting that illustrates the number of bytes in shared memory that is dynamically allocated in addition to the statically allocated memory. In this project, the *Ns* is set as same as the default value 0 and the rest is set as

lifeGame <<< *nblocks*, *512* >>> (*d_a*, *d_b*);

Where the *d_a* and *d_b* are the allocated memory on the device and *nblocks* represents the grid dimension settings to be row*column / 512 + 1.

The steps of processing the cell calculation in this model are as follows:

- Initialize two 1D arrays in the host memory (i.e. the system memory)
- Pass arrays to the GPU device global memory
- Distribute the block dimension and size for running the core function
- Call the __global__ function to process the cell calculation in each thread
- Synchronized all the threads and record the time consuming for this iteration
- Copy out the data after each iteration to the host memory, and continue running the program in the next time step until receive the interrupt signal from the user.

Overall, this section discusses the neighbor definition and calculation method in the life game and modified von Neumann CA model as well as the illustration on the parallel computing structure in CUDA platform. In the next chapter, the experiments on different platform using the same initial value would be proceeded. For the reliability of the approach, different initial values have been set in the afterwards running in a single processor and multi-processors as well.

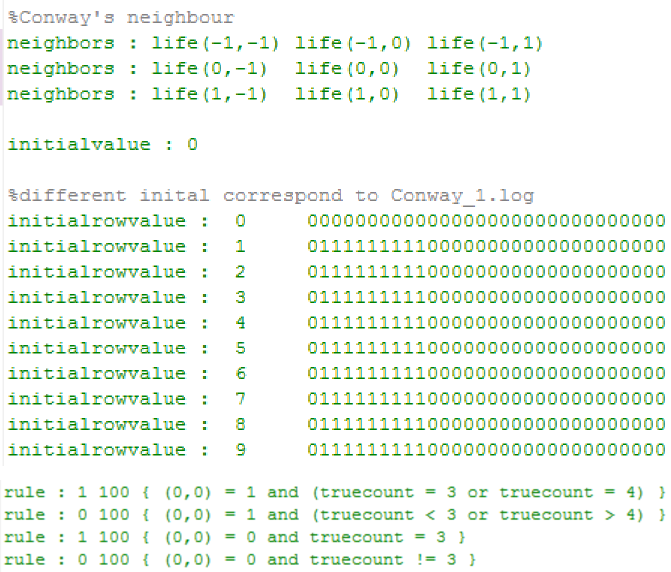
IV. SIMULATION RESULTS

In this chapter, the result obtained from three different types of computation approaches will be demonstrated and compared with each other to verify the simulation procedures and transformation. The result on two processors only appears on the efficiency analysis in Section E since the configuration of two processors' experiment only need the adjustments of the core in Visual Studio running environment parameters. In addition, the past time during the experiment is also captured for the index of efficiency in the improvement of different approaches.

A. Simulation result from CD++

The simulation setting using CD++ is demonstrated as Figure 4.1a. Different initial values are also set. Figure 4.1a to 4.1d aims to test the CD++ configuration and regard the outcomes as reference for the upcoming experiments via different approaches and platforms.

The width and length in the experiment are set to be 30 and 30, respectively, which means that there are totally 900 cells in this iteration. The rows and column could be unequal to each other definitely. The rules in the Conway life game model have been set in the transition part. Figure 4.1b to 4.1d describes the running process in CD++ through different time spot.



```
%Conway's neighbour
neighbors : life(-1,-1) life(-1,0) life(-1,1)
neighbors : life(0,-1)  life(0,0)  life(0,1)
neighbors : life(1,-1)  life(1,0)  life(1,1)

initialvalue : 0

%different inital correspond to Conway_1.log
initialrowvalue :  0     000000000000000000000000000000
initialrowvalue :  1     011111111100000000000000000000
initialrowvalue :  2     011111111100000000000000000000
initialrowvalue :  3     011111111100000000000000000000
initialrowvalue :  4     011111111100000000000000000000
initialrowvalue :  5     011111111100000000000000000000
initialrowvalue :  6     011111111100000000000000000000
initialrowvalue :  7     011111111100000000000000000000
initialrowvalue :  8     011111111100000000000000000000
initialrowvalue :  9     011111111100000000000000000000

rule : 1 100 { (0,0) = 1 and (truecount = 3 or truecount = 4) }
rule : 0 100 { (0,0) = 1 and (truecount < 3 or truecount > 4) }
rule : 1 100 { (0,0) = 0 and truecount = 3 }
rule : 0 100 { (0,0) = 0 and truecount != 3 }
```

Figure 4.1a: Initial setting on MA

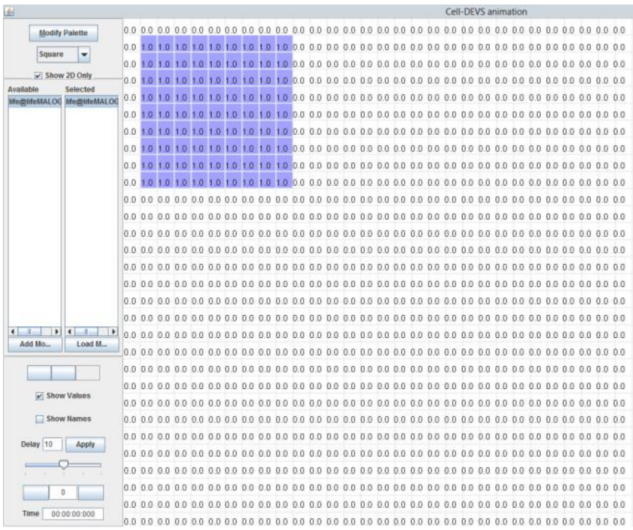


Figure 4.1b: Running in CD++ (t = 0ms)

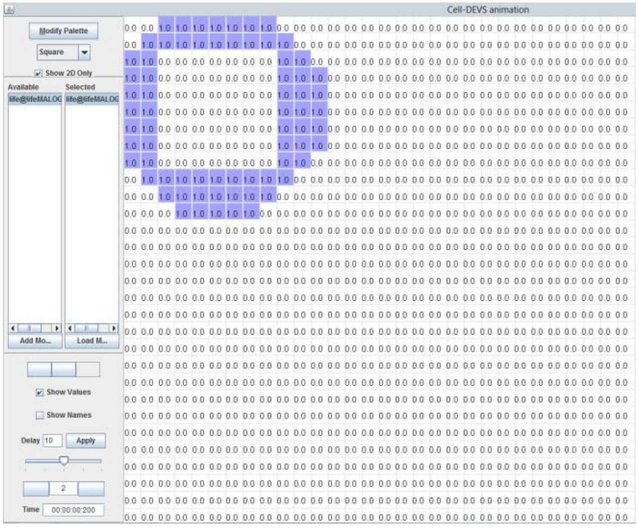


Figure 4.1c: Running in CD++ (t = 200ms)

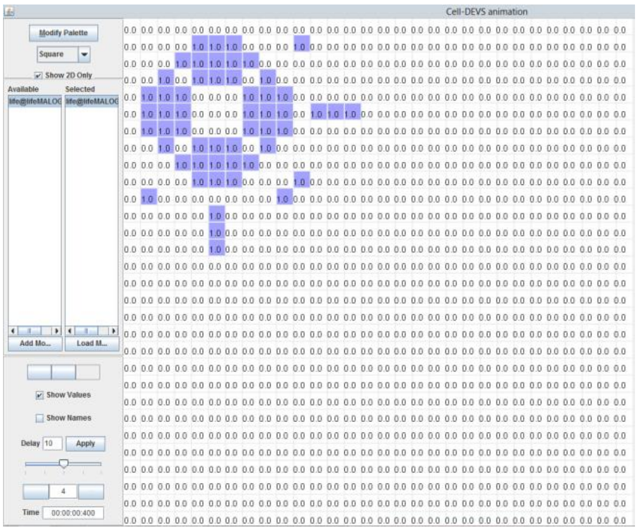
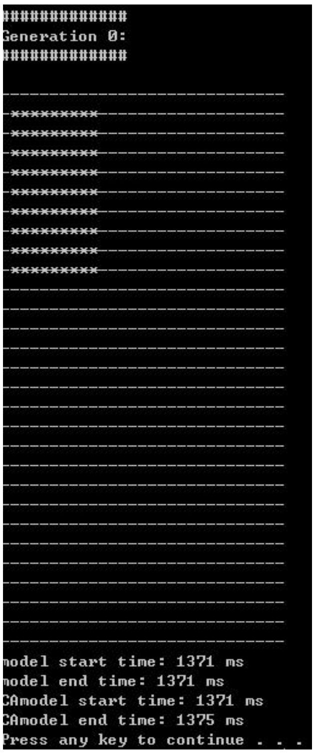


Figure 4.1d: Running in CD++ (t = 400ms)

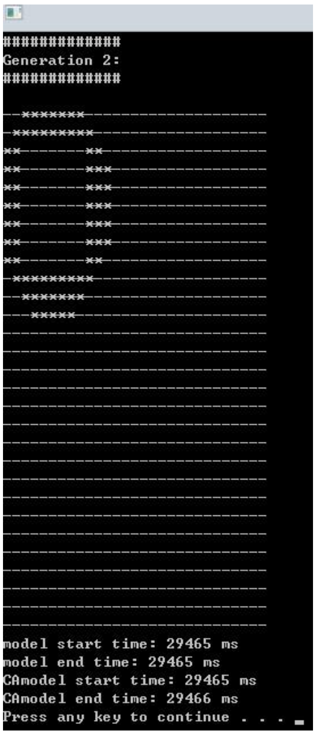
B. Simulation from single processor

Accordingly, the same initial settings have been configured in the single processing program. The simulation results for the corresponding running is shown as follows.



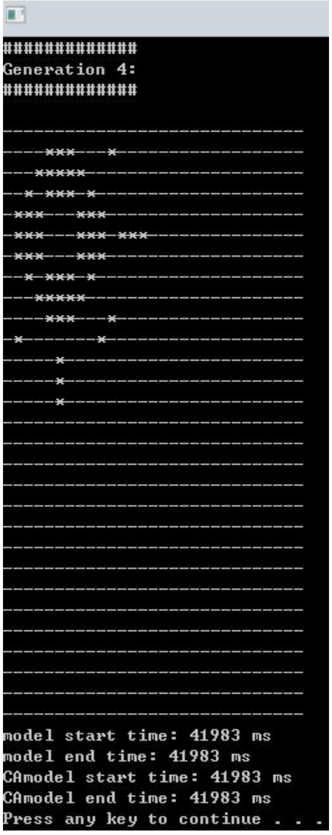
```
#############
Generation 0:
#############

------------------------------
-*********--------------------
-*********--------------------
-*********--------------------
-*********--------------------
-*********--------------------
-*********--------------------
-*********--------------------
-*********--------------------
-*********--------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
model start time: 1371 ms
model end time: 1371 ms
CAmodel start time: 1371 ms
CAmodel end time: 1375 ms
Press any key to continue . . .
```

Figure 4.2a: Running in single processor (t=0)



```
#############
Generation 2:
#############

--*******---------------------
-*********--------------------
**-------**-------------------
**-------***------------------
**-------***------------------
**-------***------------------
**-------***------------------
**-------***------------------
**-------**-------------------
-*********--------------------
--*******---------------------
---*****----------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
model start time: 29465 ms
model end time: 29465 ms
CAmodel start time: 29465 ms
CAmodel end time: 29466 ms
Press any key to continue . . .
```

Figure 4.2b: Running in single processor (t=200ms)

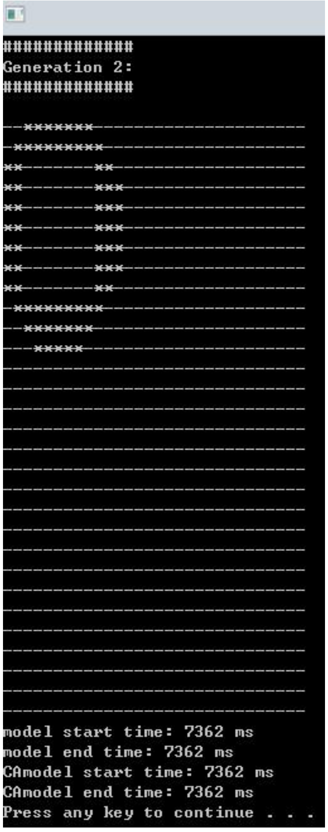


```
#############
Generation 4:
#############

------------------------------
----***---*-------------------
---*****----------------------
--*-***-*---------------------
-***---***--------------------
-***---***-***----------------
-***---***--------------------
--*-***-*---------------------
---*****----------------------
----***---*-------------------
-*-------*--------------------
-----*------------------------
-----*------------------------
-----*------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
model start time: 41983 ms
model end time: 41983 ms
CAmodel start time: 41983 ms
CAmodel end time: 41983 ms
Press any key to continue . . .
```

Figure 4.2c: Running in single processor (t=400ms)

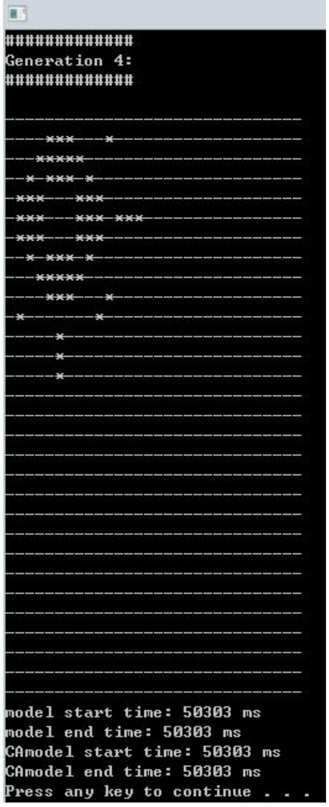
From the Figure 4.2a to 4.2c we could conclude that the results are exactly the same as the outcomes from the CD++ and in the program, a timer is used to calculate the time elapse when the program is started. The observing result suggests that the time used in generating the result is relatively small as well.

The result from the two-processor running given in below also show a confidence of stability on running the life game CA model into a two-processor program. (Figure 4.2d and Figure 4.2e)



```
#############
Generation 2:
#############

--*******---------------------
-*********--------------------
**-------**-------------------
**-------***------------------
**-------***------------------
**-------***------------------
**-------***------------------
**-------***------------------
**-------**-------------------
-*********--------------------
--*******---------------------
---*****----------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
model start time: 7362 ms
model end time: 7362 ms
CAmodel start time: 7362 ms
CAmodel end time: 7362 ms
Press any key to continue . . .
```

Figure 4.2d: Running in two-processor (t=200ms)

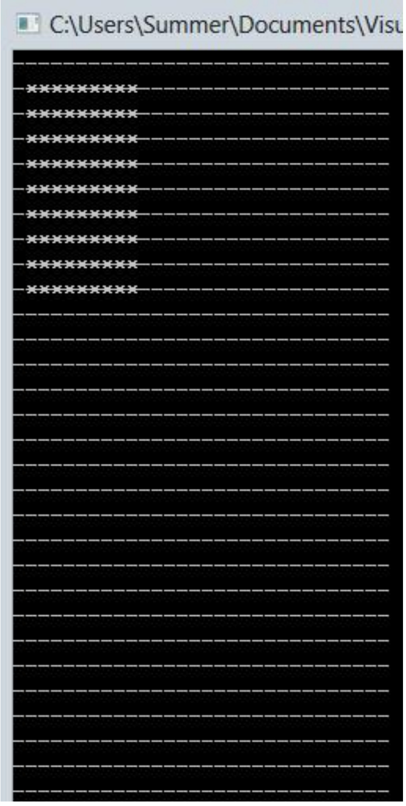


```
#############
Generation 4:
#############

------------------------------
----***---*-------------------
---*****----------------------
--*-***-*---------------------
-***---***--------------------
-***---***-***----------------
-***---***--------------------
--*-***-*---------------------
---*****----------------------
----***---*-------------------
-*-------*--------------------
-----*------------------------
-----*------------------------
-----*------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
model start time: 50303 ms
model end time: 50303 ms
CAmodel start time: 50303 ms
CAmodel end time: 50303 ms
Press any key to continue . . .
```

Figure 4.2e: Running in two-processor (t=400ms)

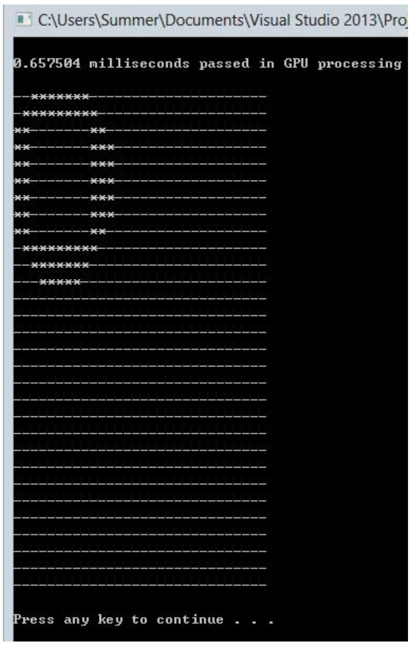
The next step is to run the same initial simulation settings on the parallel machine. The results are shown in section C in next chapter.

C. Simulation results from CUDA



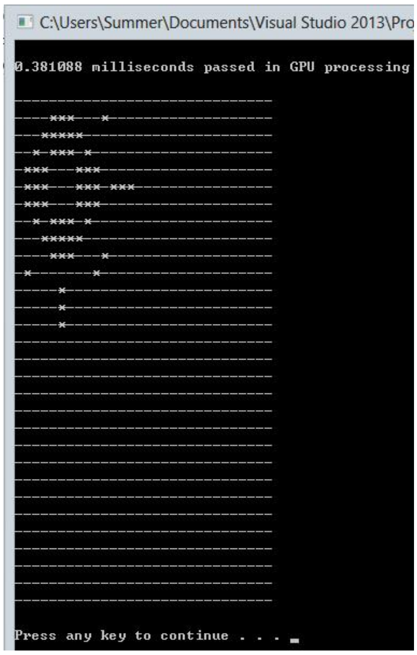
```
------------------------------
-*********--------------------
-*********--------------------
-*********--------------------
-*********--------------------
-*********--------------------
-*********--------------------
-*********--------------------
-*********--------------------
-*********--------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
```

Figure 4.3a: Running on multi-processor (t=0ms)



```
0.657504 milliseconds passed in GPU processing

--*******---------------------
-*********--------------------
**-------**-------------------
**-------***------------------
**-------***------------------
**-------***------------------
**-------***------------------
**-------***------------------
**-------**-------------------
-*********--------------------
--*******---------------------
---*****----------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------

Press any key to continue . . .
```

Figure 4.3b: Running on multi-processor (t=200ms)



```
0.381088 milliseconds passed in GPU processing

------------------------------
----***---*-------------------
---*****----------------------
--*-***-*---------------------
-***---***--------------------
-***---***-***----------------
-***---***--------------------
--*-***-*---------------------
---*****----------------------
----***---*-------------------
-*-------*--------------------
-----*------------------------
-----*------------------------
-----*------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------

Press any key to continue . . .
```

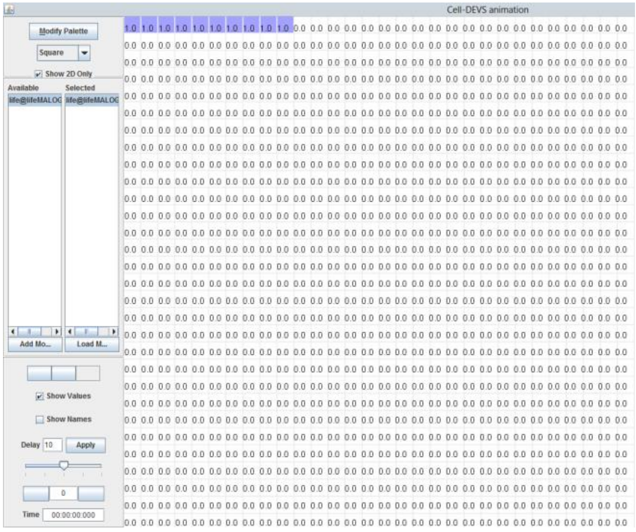
Figure 4.3c: Running on multi-processor (t=400ms)

The Figure 4.3a to 4.3c show that the simulation works properly in the parallel machine. In order to validate the model correctness, the same rules with different initial value have been set into the afterward experiments, which is demonstrated in Section D.

D. Simulation results with different initial

The programs have also experimented with different initial value to test the stability of handling the boundary conditions. The result have shown as follows.

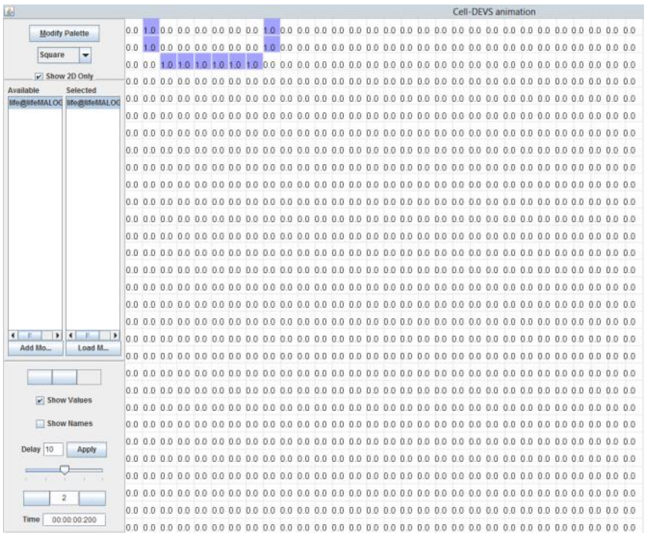
Here, the setting considers the left corner and the top boarder, in which the most common errors will be seen during the iteration.



Delay 10

Time 00:00:00:000

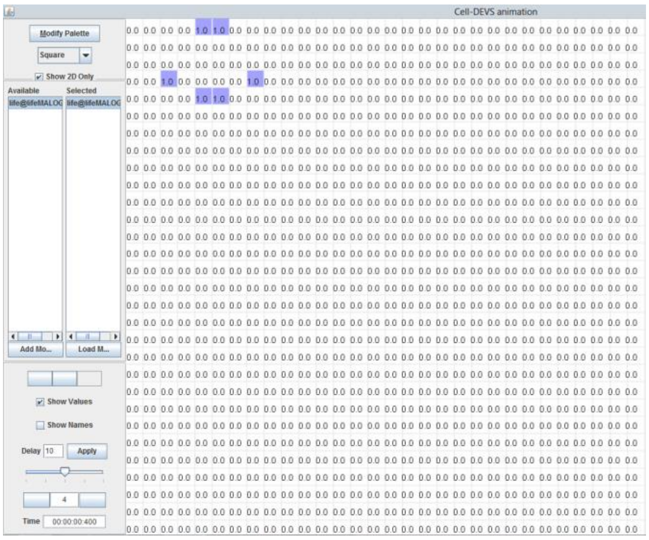
Figure 4.4a: Running different initial value in CD++ (t = 0ms)



Delay 10

Time 00:00:00:200

Figure 4.4b: Running different initial value in CD++ (t = 200ms)

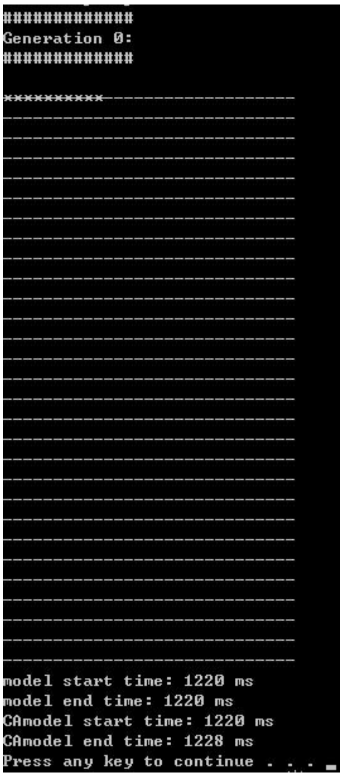


Delay 10

Time 00:00:00:400

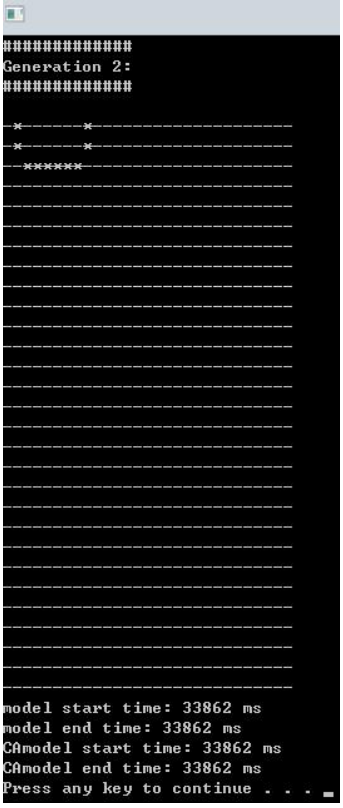
Figure 4.4c: Running different initial value in CD++ (t = 400ms)

The correspond figures shown below is running the same condition in single processor.



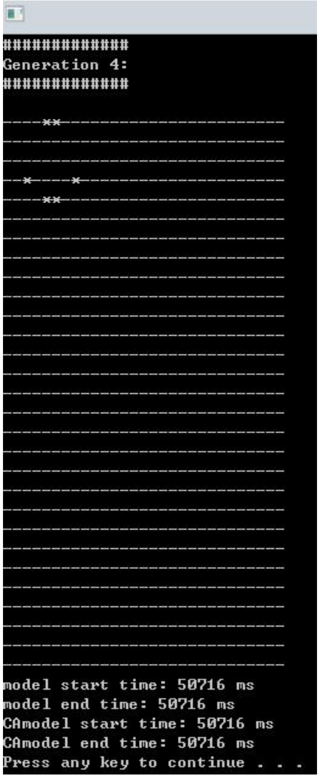
```
#############
Generation 0:
#############

**********-------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
model start time: 1220 ms
model end time: 1220 ms
CAmodel start time: 1220 ms
CAmodel end time: 1228 ms
Press any key to continue . . .
```

Figure 4.5a: Running different initial value in single processor (t = 0ms)



```
#############
Generation 2:
#############

-*------*--------------------
-*------*--------------------
--******---------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
model start time: 33862 ms
model end time: 33862 ms
CAmodel start time: 33862 ms
CAmodel end time: 33862 ms
Press any key to continue . . .
```

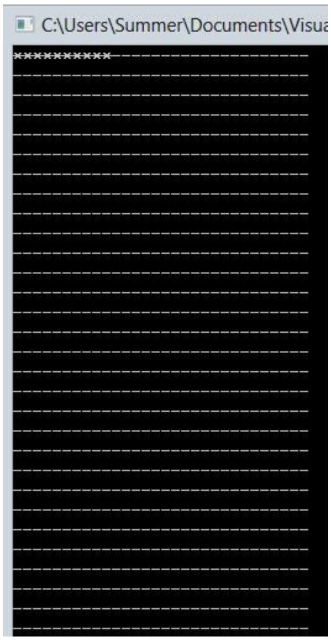
Figure 4.5b: Running different initial value in single processor (t = 200ms)



```
#############
Generation 4:
#############

----**-----------------------
-----------------------------
-----------------------------
--*----*---------------------
----**-----------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
model start time: 50716 ms
model end time: 50716 ms
CAmodel start time: 50716 ms
CAmodel end time: 50716 ms
Press any key to continue . . .
```

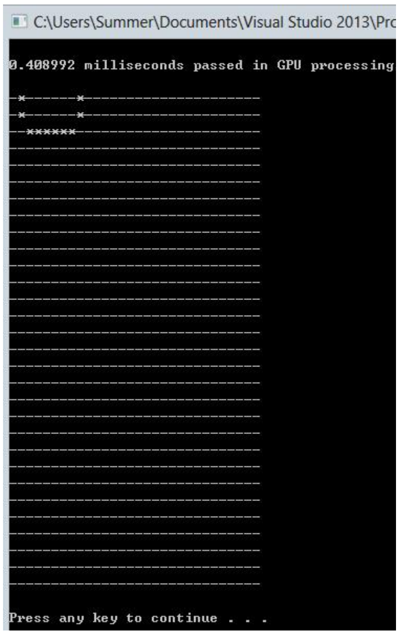
Figure 4.5c: Running different initial value in single processor (t = 400ms)

The results on multi-processing are shown as follows:



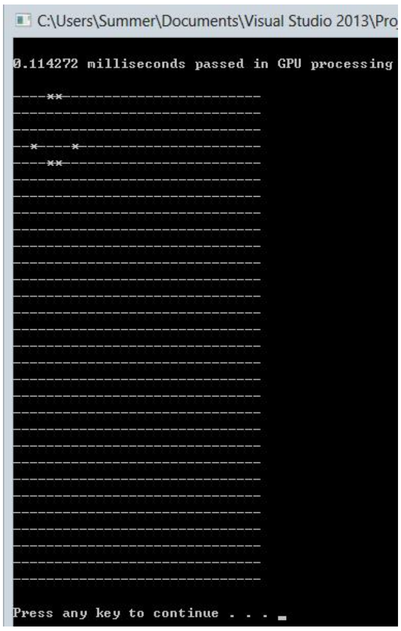
```
**********-------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
```

Figure 4.6a: Running different initial value in multi-processor (t = 0ms)



```
C:\Users\Summer\Documents\Visual Studio 2013\Pro

0.408992 milliseconds passed in GPU processing

-*------*---------------------
-*------*---------------------
--******----------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------

Press any key to continue . . .
```

Figure 4.6b: Running different initial value in multi-processor (t = 200ms)



```
C:\Users\Summer\Documents\Visual Studio 2013\Pro

0.114272 milliseconds passed in GPU processing

----**------------------------
------------------------------
------------------------------
--*----*----------------------
----**------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------

Press any key to continue . . . _
```

Figure 4.6c: Running different initial value in multi-processor (t = 400ms)

The figures below is from the other test case. In this test case, arbitrary distributions on the initial value are given at the first place. The running also proves that the function of the system can work normally.

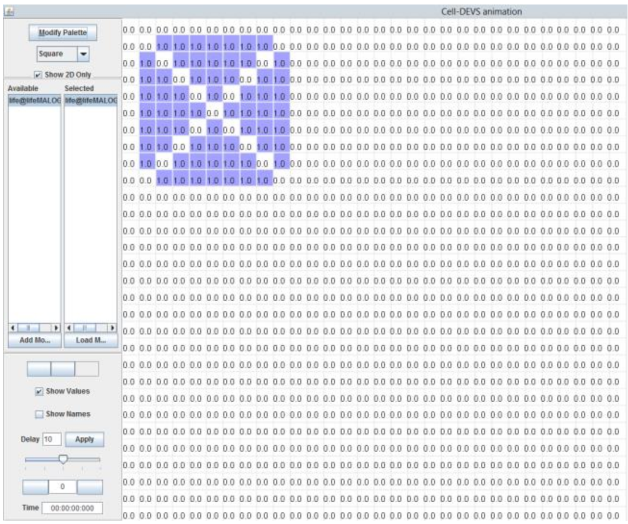


Figure 4.7a: Running different initial value in CD++ (t = 0ms)

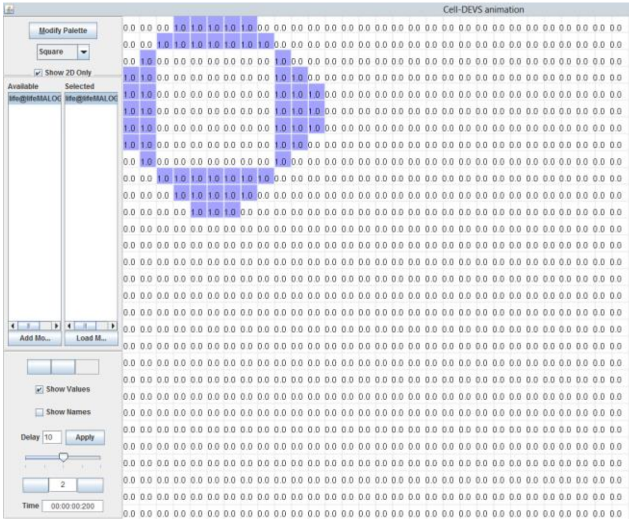


Figure 4.7b: Running different initial value in CD++ (t = 200ms)

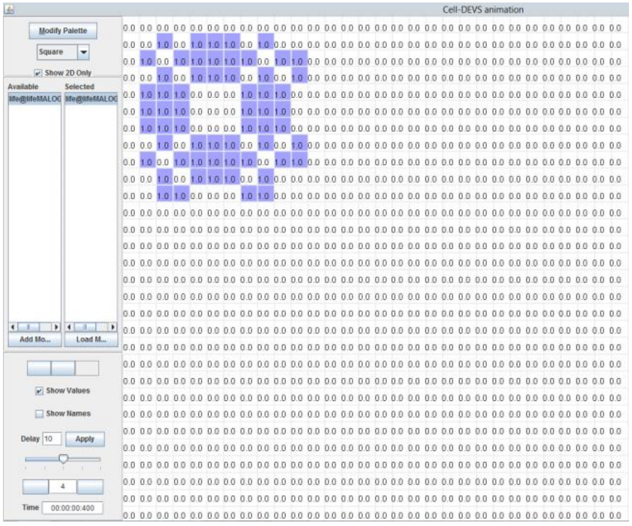
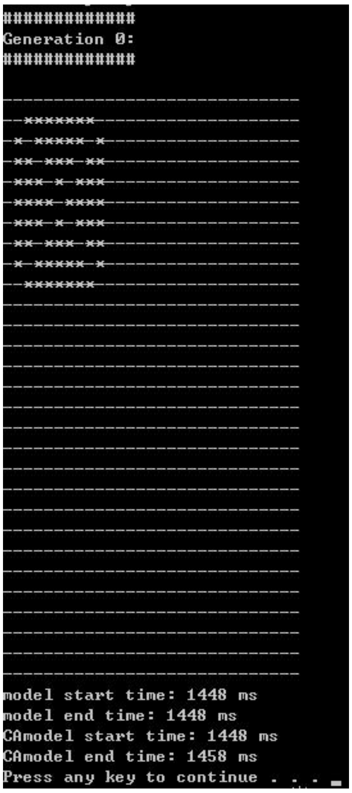
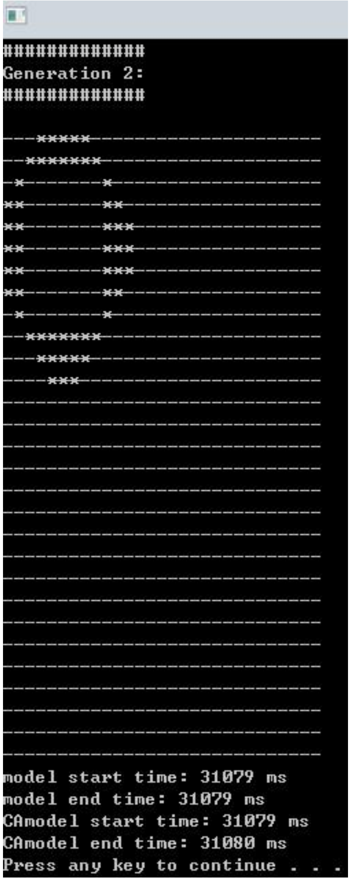


Figure 4.7c: Running different initial value in CD++ (t = 400ms)



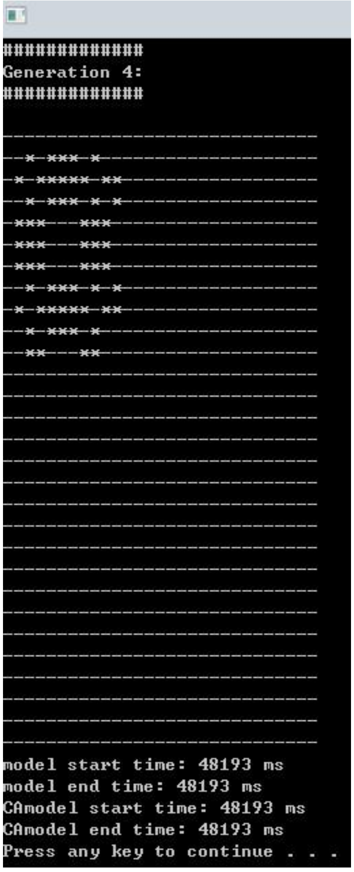
```
#############
Generation 0:
#############

------------------------------
--*******---------------------
-*-*****-*--------------------
-**-***-**--------------------
-***-*-***--------------------
-****-****--------------------
-***-*-***--------------------
-**-***-**--------------------
-*-*****-*--------------------
--*******---------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
model start time: 1448 ms
model end time: 1448 ms
CAmodel start time: 1448 ms
CAmodel end time: 1458 ms
Press any key to continue . . .
```

Figure 4.8a: Running different initial value in single processor (t =0ms)



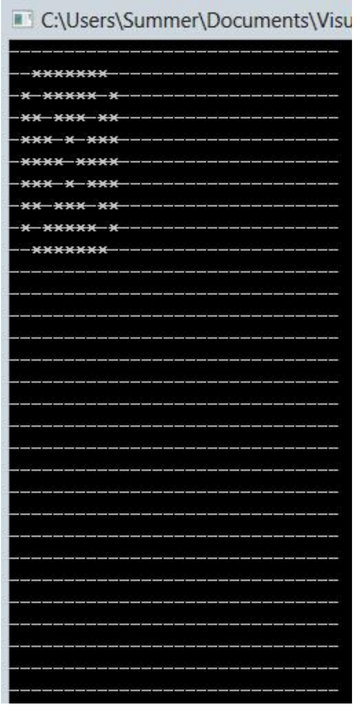
```
#############
Generation 2:
#############

---*****----------------------
--*******---------------------
-*-------*--------------------
**-------**-------------------
**-------***------------------
**-------***------------------
**-------***------------------
**-------**-------------------
-*-------*--------------------
--*******---------------------
---*****----------------------
----***-----------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
model start time: 31079 ms
model end time: 31079 ms
CAmodel start time: 31079 ms
CAmodel end time: 31080 ms
Press any key to continue . . .
```

Figure 4.8b: Running different initial value in single processor (t =200ms)



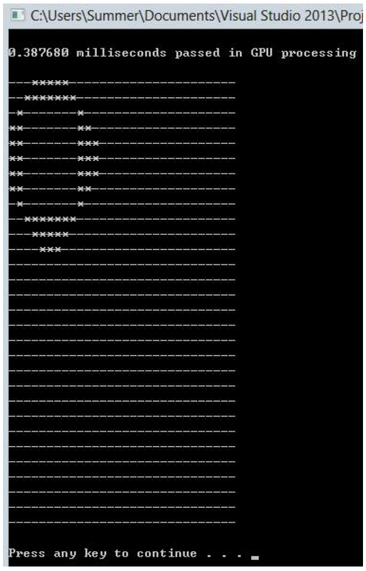
```
#############
Generation 4:
#############

------------------------------
--*-***-*---------------------
-*-*****-**-------------------
--*-***-*-*-------------------
-***---***--------------------
-***---***--------------------
-***---***--------------------
--*-***-*-*-------------------
-*-*****-**-------------------
--*-***-*---------------------
--**---**---------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
model start time: 48193 ms
model end time: 48193 ms
CAmodel start time: 48193 ms
CAmodel end time: 48193 ms
Press any key to continue . . .
```

Figure 4.8c: Running different initial value in single processor (t =400ms)



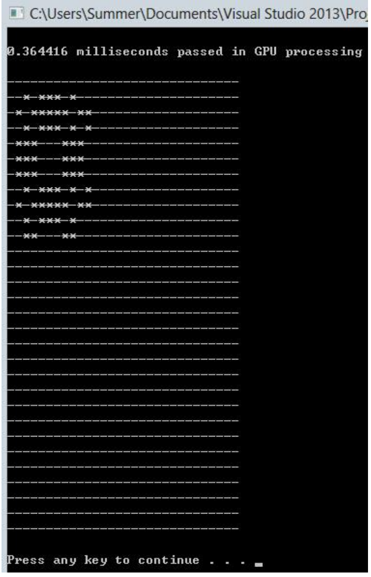
```
------------------------------
--*******---------------------
-*-*****-*--------------------
-**-***-**--------------------
-***-*-***--------------------
-****-****--------------------
-***-*-***--------------------
-**-***-**--------------------
-*-*****-*--------------------
--*******---------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
```

Figure 4.9a: Running different initial value in multi-processor (t =0ms)



```
0.387680 milliseconds passed in GPU processing

---*****----------------------
--*******---------------------
-*-------*--------------------
**-------**-------------------
**-------***------------------
**-------***------------------
**-------***------------------
**-------**-------------------
-*-------*--------------------
--*******---------------------
---*****----------------------
----***-----------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------

Press any key to continue . . .
```

Figure 4.9b: Running different initial value in multi-processor (t =200ms)



```
0.364416 milliseconds passed in GPU processing

------------------------------
--*-***-*---------------------
-*-*****-**-------------------
--*-***-*-*-------------------
-***---***--------------------
-***---***--------------------
-***---***--------------------
--*-***-*-*-------------------
-*-*****-**-------------------
--*-***-*---------------------
--**---**---------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------

Press any key to continue . . .
```

Figure 4.9c: Running different initial value in multi-processor (t =400ms)

E. Large scale simulation

In order to test the reliability in large elements surroundings, the tests have included the scale of N2 which is upmost to 100 million elements in single processing and two-processor processing. Yet, for the simplicity of the calculation, the rows and columns are set to be the same. This test is for showing the improvement of the multi-processor. There will be total number of scale (N2) in the actual running. In addition, the used initial value is generated by the rand() function. The table 4.1 to 4.3 show the outcomes from different circumstance.

Figure 4.10 describes that the calculation time in the corresponding scales. It indicates that the dramatic reduction of the time consuming in the CUDA is lesser than running in single processor or two-processor, which is similar to the expectation.

Scale(N)	1st(ms)	2nd(ms)	3rd(ms)	4th(ms)	Avg.(ms)
100	0	0	0	0	0
500	7	5	6	5	5.6
1000	16	23	23	22	21.2
3000	230	209	199	197	205.6
5000	652	564	557	548	572.2
7000	1332	1136	1086	1079	1141.4
9000	2081	1856	1805	1830	1864.6
10000	2606	2261	2225	2250	2313.4
12000	3739	3281	3190	3198	3316

Table 4.1: Simulation result from single processor

Scale(N)	1st(ms)	2nd(ms)	3rd(ms)	4th(ms)	Avg.(ms)
100	0	0	0	0	0
500	4	3	4	4	3.75
1000	15	15	15	14	14.75
3000	127	126	126	124	125.75
5000	358	353	352	351	353.5
7000	687	695	689	736	701.75
9000	1152	1145	1132	1149	1144.5
10000	1428	1420	1425	1471	1436

Table 4.2: Simulation result from 2 processors on OpenMP

Scale (N)	1st(ms)	2nd(ms)	3rd(ms)	4th(ms)	Avg.(ms)
100	0.0862	0.3803	0.4018	0.376	0.3111
500	0.7504	1.0238	1.0505	1.0195	0.9610
1000	2.3312	2.6598	2.6461	2.6216	2.5647
3000	18.961	19.376	19.401	19.317	19.264
5000	51.497	51.497	51.497	51.497	51.497
5500	62.364	63.014	62.904	62.832	62.779

Table 4.3: Simulation result from multi-processors on CUDA

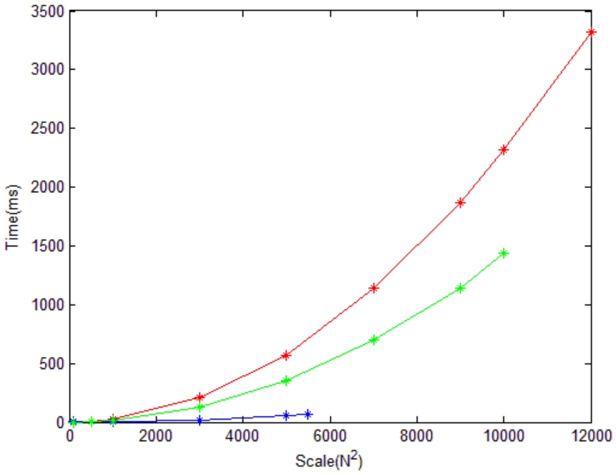


Figure 4.10: Time consuming through different scale in single processor (red line), 2 processors (green line) and multi-processors (blue line)

F. *Other types of CA model—modified von Neumann CA model*

By changing the transition rule and neighborhood definition, a different CA model could be built as follows.

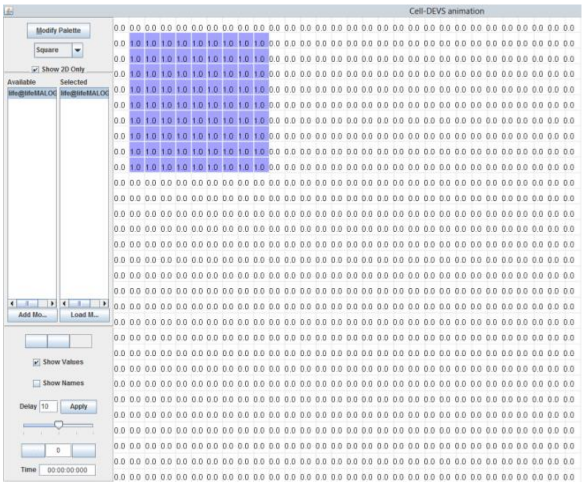


Figure 4.11a: Modified Neumann's CA model (t=0ms)

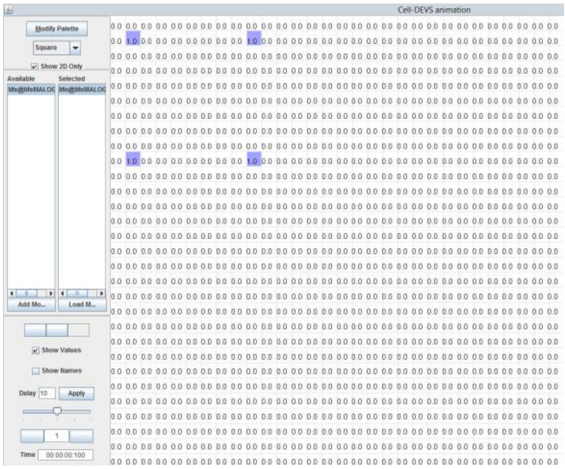


Figure 4.11b: Modified Neumann's CA model (t=100ms)

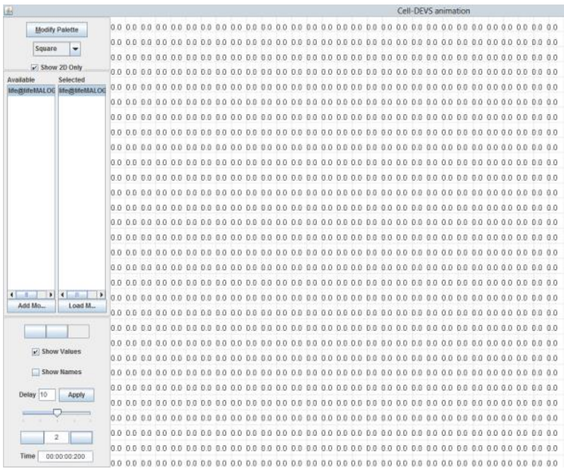
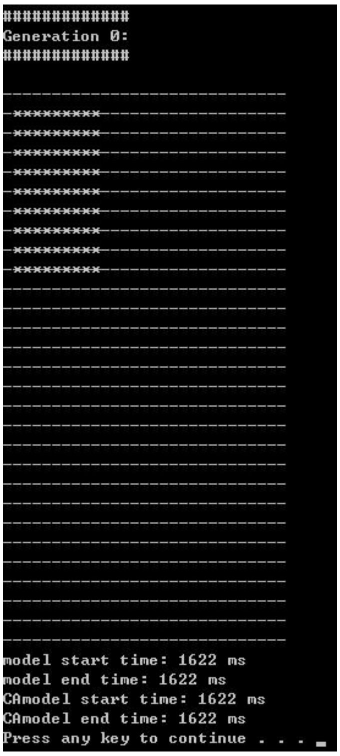
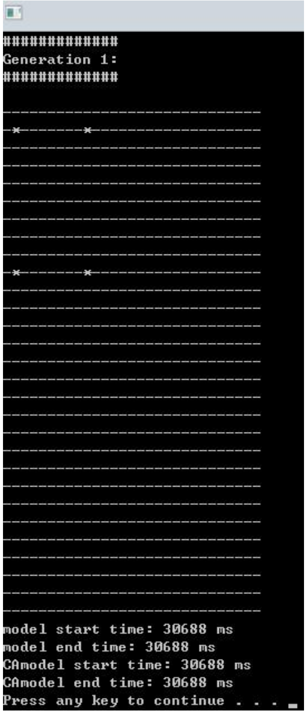


Figure 4.11c: Modified Neumann's CA model (t=200ms)



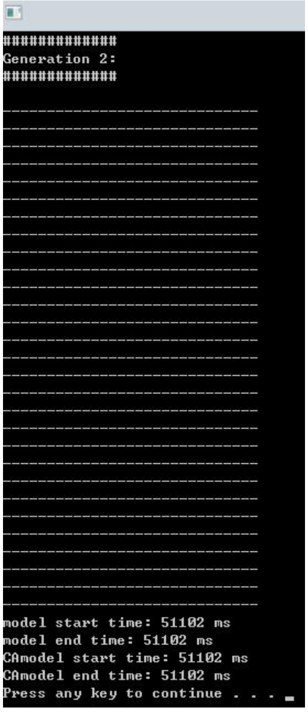
```
#############
Generation 0:
#############

-----------------------------
-*********-------------------
-*********-------------------
-*********-------------------
-*********-------------------
-*********-------------------
-*********-------------------
-*********-------------------
-*********-------------------
-*********-------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
model start time: 1622 ms
model end time: 1622 ms
CAmodel start time: 1622 ms
CAmodel end time: 1622 ms
Press any key to continue . . .
```

Figure 4.12a: Modified Neumann's CA model running in single processor (t=0ms)



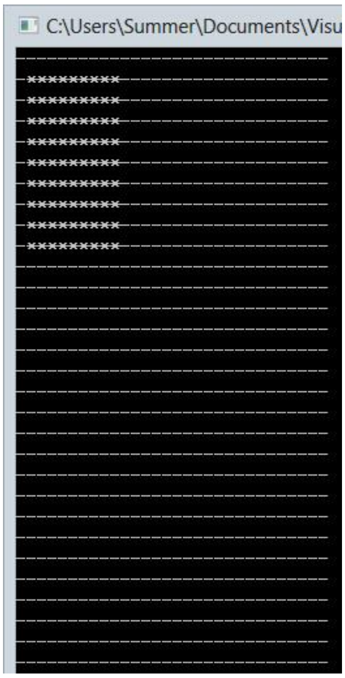
```
#############
Generation 1:
#############

-----------------------------
-*-------*-------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-*-------*-------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
model start time: 30688 ms
model end time: 30688 ms
CAmodel start time: 30688 ms
CAmodel end time: 30688 ms
Press any key to continue . . .
```

Figure 4.12b: Modified Neumann's CA model running in single processor (t=100ms)



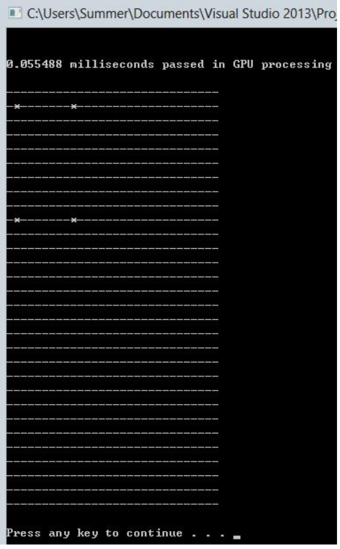
```
#############
Generation 2:
#############

-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
model start time: 51102 ms
model end time: 51102 ms
CAmodel start time: 51102 ms
CAmodel end time: 51102 ms
Press any key to continue . . .
```

Figure 4.12c: Modified Neumann's CA model running in single processor (t=200ms)



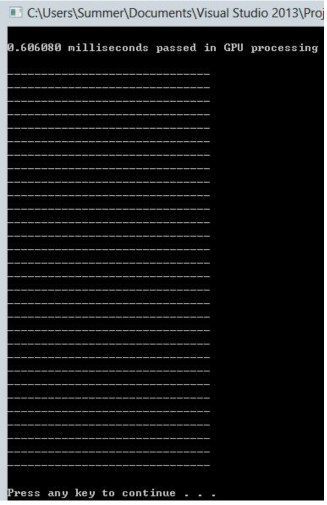
```
-----------------------------
-*********-------------------
-*********-------------------
-*********-------------------
-*********-------------------
-*********-------------------
-*********-------------------
-*********-------------------
-*********-------------------
-*********-------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
```

Figure 4.13a: Modified Neumann's CA model running in mult-processor (t=0ms)



```

0.055488 milliseconds passed in GPU processing

-----------------------------
-*-------*-------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-*-------*-------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------

Press any key to continue . . .
```

Figure 4.13b: Modified Neumann's CA model running in mult-processor (t=100ms)



```
0.606080 milliseconds passed in GPU processing

------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------

Press any key to continue . . .
```

Figure 4.13c: Modified Neumann’s CA model running in mult-processor (t=200ms)

V. CONCLUSIONS

From the various experiment on Conway life game, the evidence shows that the calculation time on the given problem domain has reduced by applying the parallel solving method the cellular automata model. Moreover, another type of CA model such as the modified von Neumann CA model has been used to validate the stability of the parallel approach as well.

Besides, the performance applying parallel computing with CUDA in the Cellular Automata model have a remarkable advance comparing to the single and two-processor in which the single processor have an exponential increasing while the two-processor approach only have a slightly slow but also the same trend as the single processor.

ACKNOWLEDGMENT

I would like to thank Dr. Gabriel A. Wainer for his patient guidance and helpful advice during my project and assignments, and for introducing me the realm of modeling and simulation of discrete events.

I would like to thank Sixuan Wang and my classmates for their helps in my assignments when I meet the bottle neck of my assignments as well.

REFERENCES

[1] Cellularr automaton ,Wiki, http://en.wikipedia.org/wiki/Cellular_automaton, 2014

[2] Chowdhury, D.R.;Basu, S.;Gupta, I.S.;Chaudhuri, P.P., “Design of CAECC - cellular automata based error correcting code,” IEEE Transactions, vol. 43(6), pp. 759-764, June 1994

[3] Parallel programming model, Wiki, http://en.wikipedia.org/wiki/Parallel_programming_model, 2014

[4] CUDA, Wiki, http://en.wikipedia.org/wiki/CUDA, 2014

[5] CUDA Tutorial, http://geco.mines.edu/tesla/cuda_tutorial_mio/, 2014

[6] Rybacki, S.; Himmelspach, J.Experiments with Single Core, Multi-core, and GPU Based Computation of Cellular Automata, Advances in System Simulation, 2009. SIMUL '09. pp.62-67, Sept 2009

[7] NVIDIA, “What is cuda,” http://www.nvidia.com/object/cuda what is. html, 16.01.2009.

[8] AMD/ATI, “Gpu technology for accelerated computing,” http://www.amd.com/stream, 16.01.2009.

[9] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron, “A performance study of general-purpose applications on graphics processors using cuda,” J. Parallel Distrib. Comput., vol. 68, no. 10, pp. 1370–1380, 2008.

[10] L. Nyland, M. Harris, and J. Prins, “Fast n-body simulation with cuda,” in GPU Gems 3, H. Nguyen, Ed. Addison Wesley Professional, August 2007, ch. 31.

[11] W. R. Mark, R. S. Glanville, K. Akeley, and M. J. Kilgard, “Cg: a systemfor programming graphics hardware in a c-like language,” in SIGGRAPH03: ACM SIGGRAPH 2003 Papers. New York, NY, USA: ACM, 2003,pp. 896–907.

[12] Microsoft, “Writing hlsl shaders in direct3d 9,” http://msdn.microsoft.com/en-us/library/bb944006(VS.85).aspx,16.01.2009.

[13] J. Nickolls, I. Buck, M. Garland, and K. Skadron, “Scalable parallel programming with cuda,” Queue, vol. 6, no. 2, pp. 40–53, 2008.

[14] J. Himmelspach and A. M. Uhrmacher, “Plug’n simulate,” in ANSS ’07:Proceedings of the 40th Annual Simulation Symposium. Washington,DC, USA: IEEE Computer Society, Mar. 2007, pp. 137–143.

[15] D. Johnson, “A theoretician’s guide to the experimental analysis of algorithms,” in Fifth and Sixth DIMACS Implentation Challenges, 2002.

[16] S. Gobron, F. Devillard, and B. Heit, “Retina simulation using cellular automata and gpu programming,” Mach. Vision Appl., vol. 18, no. 6, pp.331–342, 2007.

[17] M. Corporation, Microsoft DirectX 9 Programmable Graphics Pipeline. Redmond, WA, USA: Microsoft Press, 2003.

[18] S. Druon, A. Crosnier, and L. Brigandat, “Efficient cellular automata for 2d/3d free-form modeling,” in In WSCG, 2003, p. 102108.