

Parallel and Distributed Systems

Complete Report

Student ID: st20240706

Mananduwa Acharige, Chamath Shyamal



Acknowledgement

Primarily, I take this opportunity to express my deep sense of gratitude and my profound respect to the lecturer who guided and inspired me in doing this assignment. I had to get the guidance of a respected and responsible person at the preparation time and while continuing the coding and the documentation. So, I would like to thank our lecturer Mr. Paul Angel whose valuable guidelines and consultations been the ones that helped me patch this assignment and make this finalize and full proof success.

Also, his instructions which were given underlying the structure has served as the major contributor towards in completing this assignment and his instructions about the programming and coding was more helpful in implementing both the Sequential and Parallel Versions. Thanks to them and this golden opportunity, I got the full knowledge on advanced C++ Programming and Algorithms whereas I could complete only the basics in the First Year of this degree. I hope this knowledge you gave me will help me in learning more advance C++ programming, complex algorithms as well as my career. I really thankful for you because of the massive courage you had to teach us.

Then I would like to thank Mr. Rehmat Ullah, my batch mates and other lecturers who are with us and help us in many sides since the beginning of the Program. However, I am really grateful because I managed to complete this assessment within the time period given by Mr. Paul.

Thank you all!

Contents

Introduction.....	3
1 – Patterns Implemented	4
1.1 – Pattern 1 (Dead).....	4
1.2 – Pattern 2 - Breadcrumb Grenade (Dead).....	5
1.3 – Pattern 3 – Pulsar (Repeat)	6
1.4 – Pattern 4 (Repeat).....	7
2 – Experiment 1 (Comparing the Sequential Version with Parallel Version)	8
2.1– Plan 1 (Testing the Sequential Version with Pattern 1 -Dead)	9
Test 1 (Same Pattern, First Execution) :.....	9
Test 2 (Same Pattern, Second Execution) :	10
Test 3 (Same Pattern, Third Execution) :	10
2.2 – Plan 2 (Testing the Parallel Version with Pattern 1 -Dead).....	12
Test 1 (Same Pattern, First Execution) :.....	12
Test 2 (Same Pattern, Second Execution) :	12
Test 3 (Same Pattern, Third Execution) :	13
Comparison	14
3 – Experiment 2 (Varying the Array Size)	15
3.1 - Plan1 (Array Size = 50 x 50).....	15
3.2 - Plan2 (Array Size = 100 x 100).....	16
3.3 - Plan3 (Array Size = 1000 x 1000).....	17
.....	17
Comparison	18
4 – Analysis Made from Experiment Outcomes	19
Conclusion.....	20
References	21

Introduction

Nowadays, there are massive amount of data being produced and it has become crucial to implement competent algorithms to handle those data. This paper aims to outline a set of experiments to run against the two implementations (Sequential & TBB Parallel) with regard to John Conway created cellular automaton known as "Life," often known as the "Game of Life." Although the underlying ideas are simple in it, the system's emergent behaviour may be complex and unexpected.

Therefore, this paper demonstrates understanding of the theoretical concepts and abstractions to the design of novel and innovative distributed and parallel systems. Also, this evaluates the fundamental issues in the design of distributed algorithms, protocols and systems, such as timing, coordination and consensus. Basically, this is just like a testing phase conducted to check the performance of the two algorithms built to implement the Conway's Game of Life rules which runs in Visual Studio IDE. The Sequential Version is a classical implementation to implement the four rules of the Conway's Game of Life whereas the Parallel Version takes the benefits of multiple CPU cores to provide outputs within a shorter period of time.

In this study, its specifics both algorithms, including their designs, implementations, and performance traits. Additionally, it contrasts the effectiveness of sequential and parallel algorithms on various parameters.

Hope this assessment will be a clear and ideal one.

1 – Patterns Implemented

1.1 – Pattern 1 (Dead)

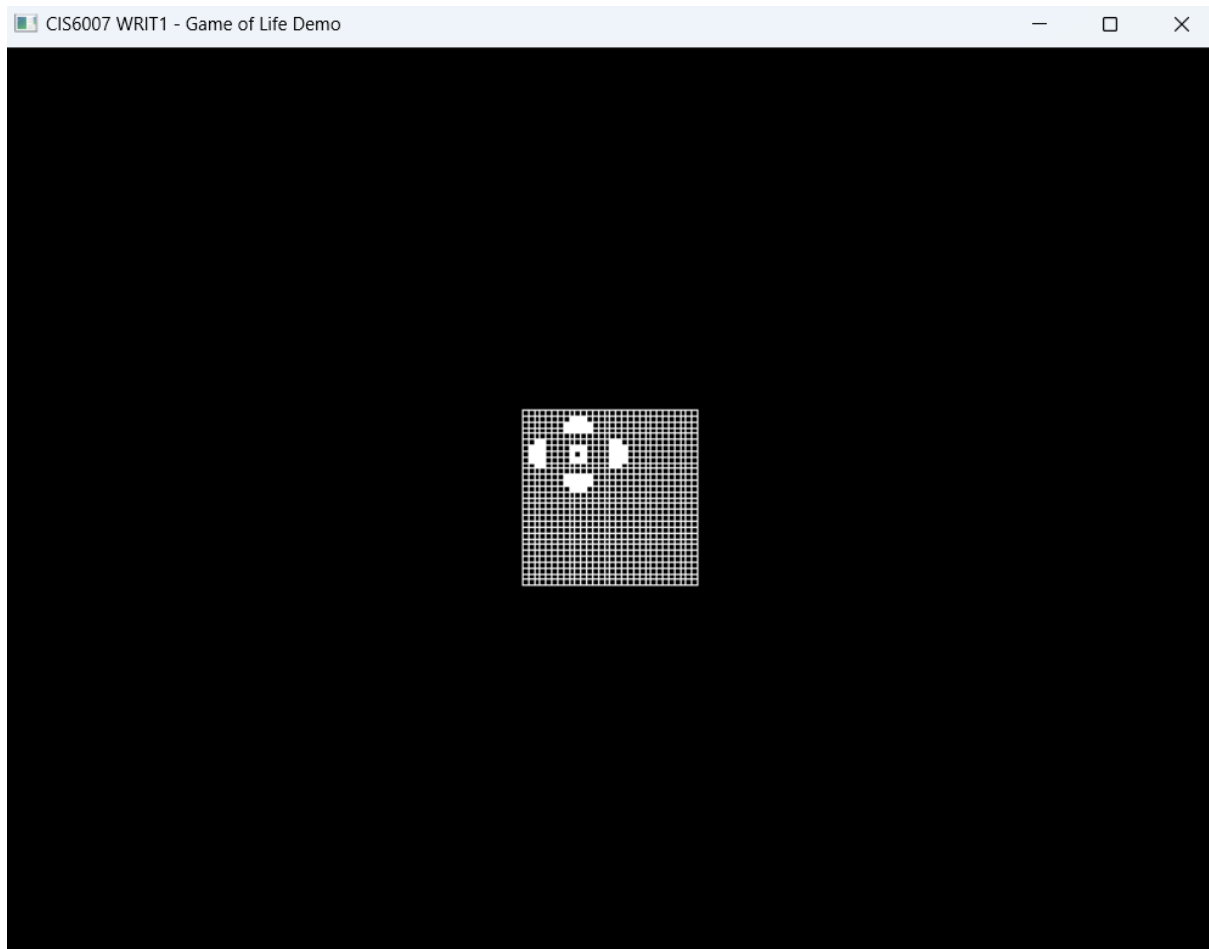


Figure 1

1.2 – Pattern 2 - Breadcrumb Grenade (Dead)

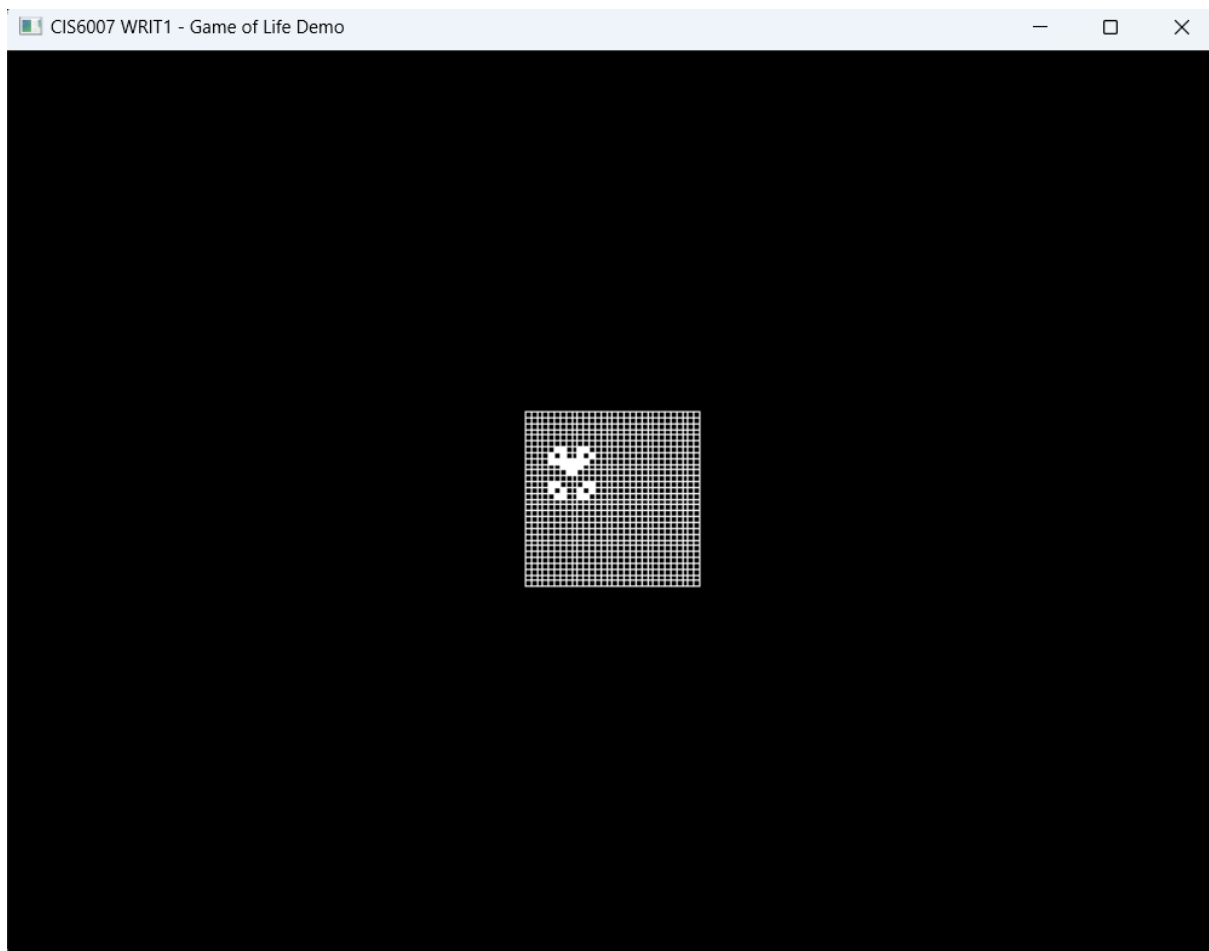


Figure 2

1.3 – Pattern 3 – Pulsar (Repeat)

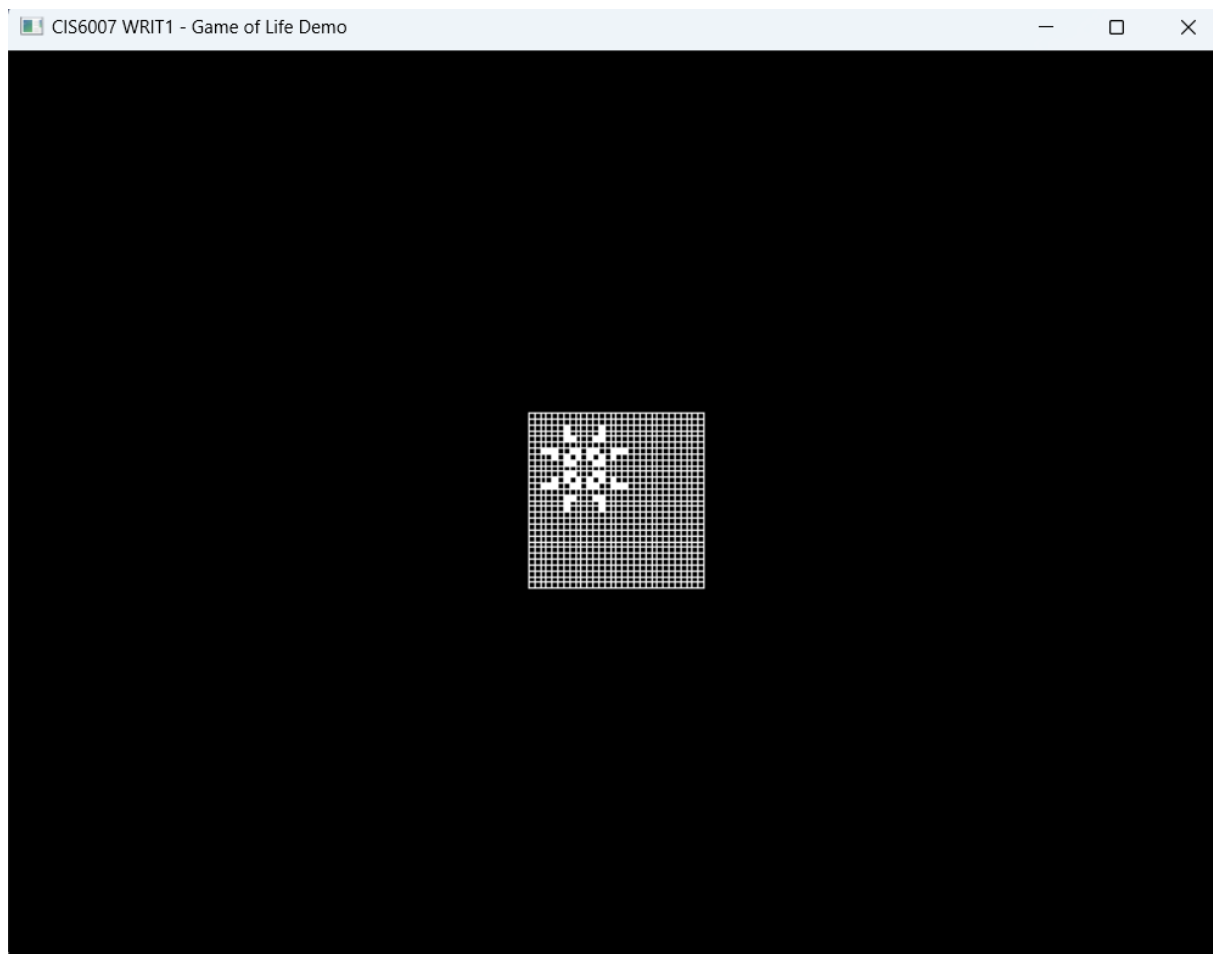


Figure 3

1.4 – Pattern 4 (Repeat)

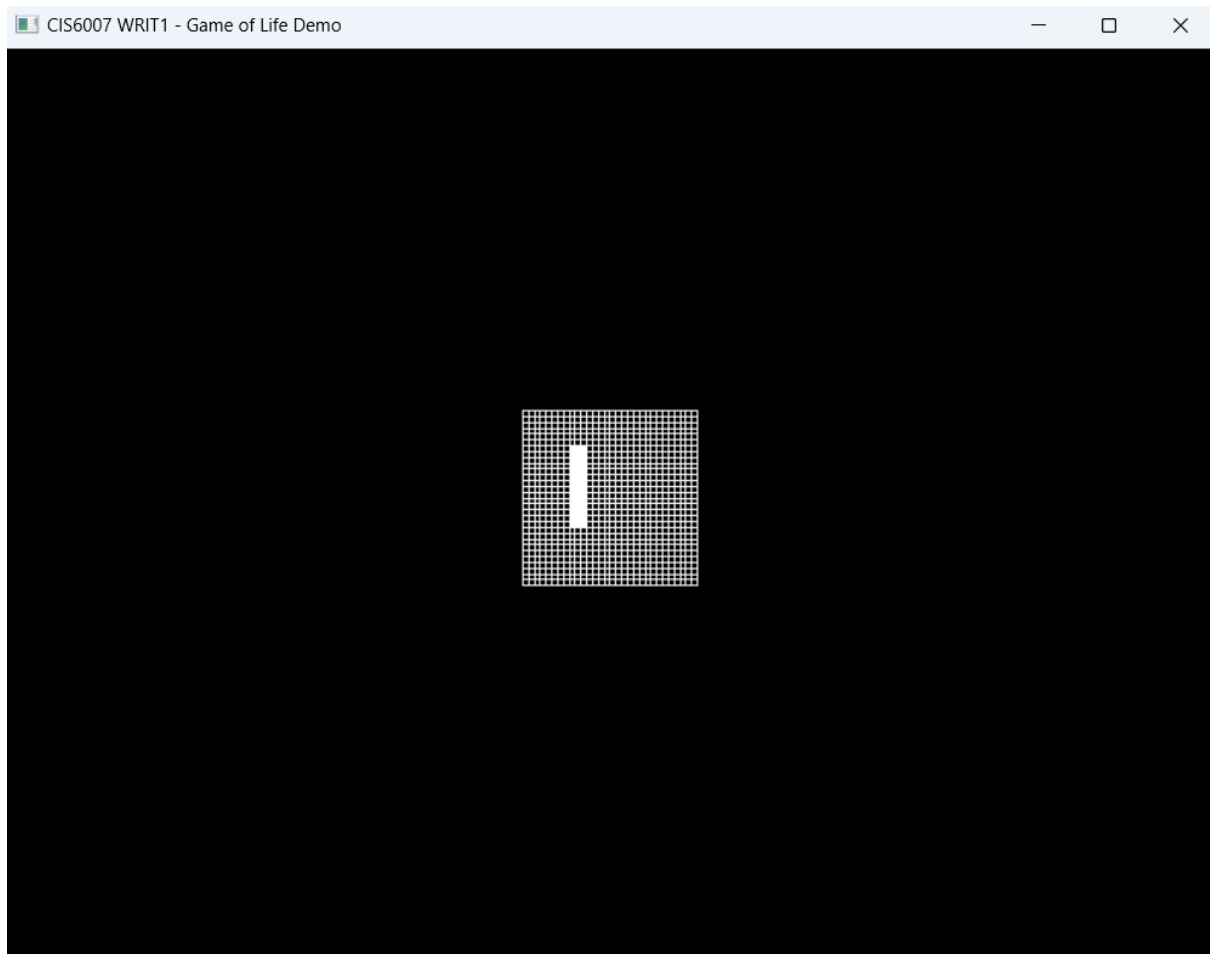
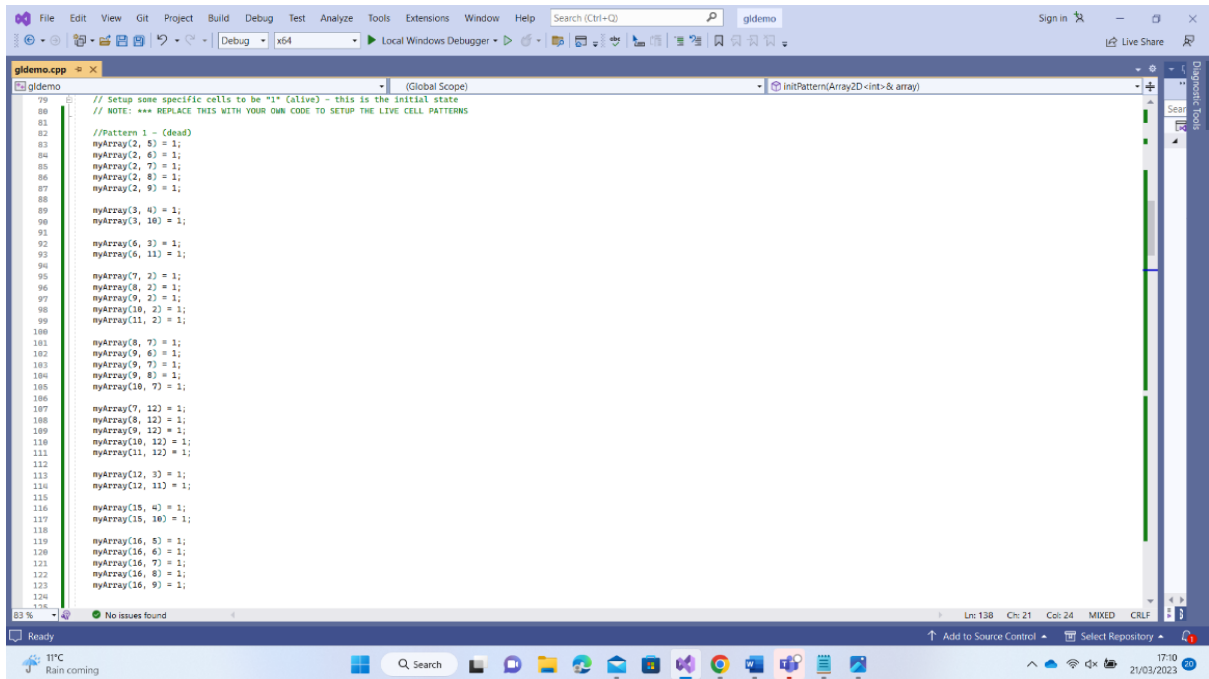


Figure 4

2 – Experiment 1 (Comparing the Sequential Version with Parallel Version)

Below depicts the Pattern which will be used for this experiment:



```
79 // Setup some specific cells to be "1" (Alive) - this is the initial state
80 // NOTE: *** REPLACE THIS WITH YOUR OWN CODE TO SETUP THE LIVE CELL PATTERNS
81
82 //Pattern 1 - (dead)
83 myArray(2, 5) = 1;
84 myArray(2, 6) = 1;
85 myArray(2, 7) = 1;
86 myArray(2, 8) = 1;
87 myArray(2, 9) = 1;
88
89 myArray(3, 4) = 1;
90 myArray(3, 10) = 1;
91
92 myArray(6, 3) = 1;
93 myArray(6, 11) = 1;
94
95 myArray(7, 2) = 1;
96 myArray(8, 2) = 1;
97 myArray(9, 2) = 1;
98 myArray(10, 2) = 1;
99 myArray(11, 2) = 1;
100
101 myArray(8, 7) = 1;
102 myArray(9, 6) = 1;
103 myArray(9, 7) = 1;
104 myArray(9, 8) = 1;
105 myArray(10, 7) = 1;
106
107 myArray(7, 12) = 1;
108 myArray(8, 12) = 1;
109 myArray(9, 12) = 1;
110 myArray(10, 12) = 1;
111 myArray(11, 12) = 1;
112
113 myArray(12, 3) = 1;
114 myArray(12, 11) = 1;
115
116 myArray(15, 4) = 1;
117 myArray(15, 10) = 1;
118
119 myArray(16, 5) = 1;
120 myArray(16, 6) = 1;
121 myArray(16, 7) = 1;
122 myArray(16, 8) = 1;
123 myArray(16, 9) = 1;
```

Figure 5

Below depicts the graphical representation of the above implemented Pattern:

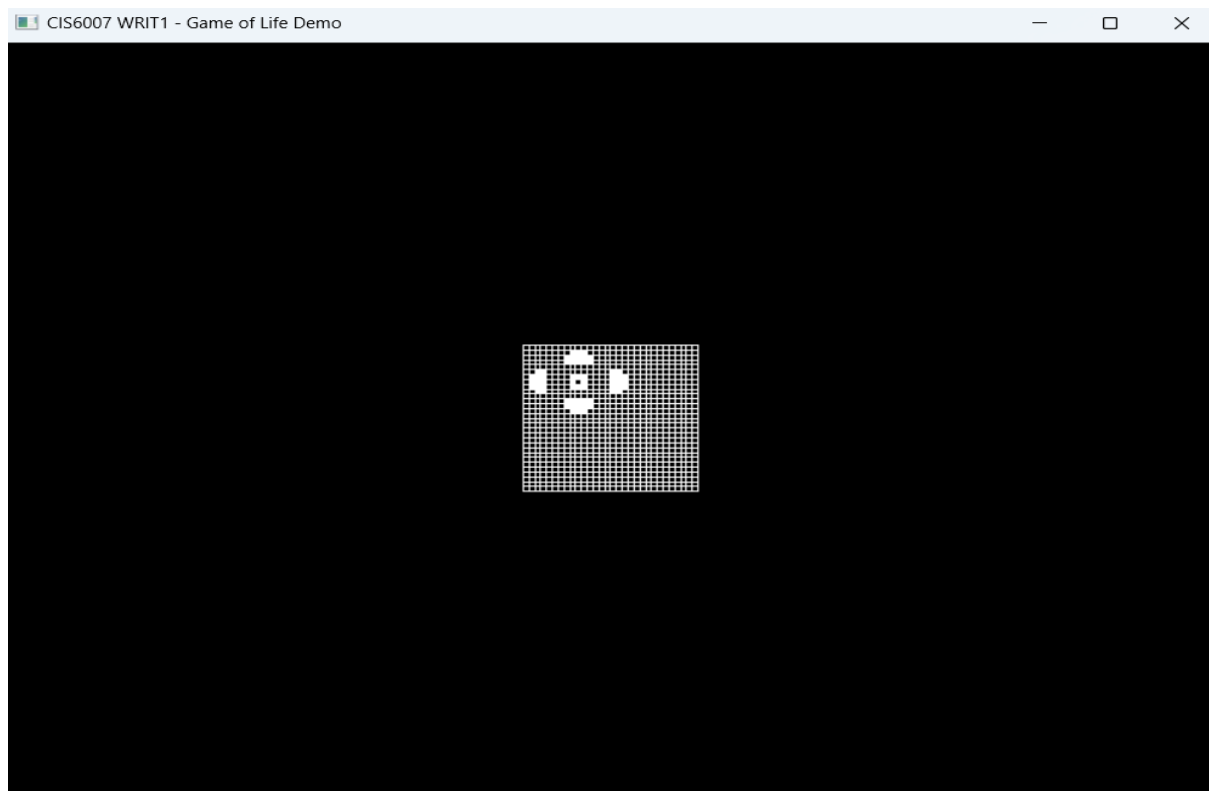
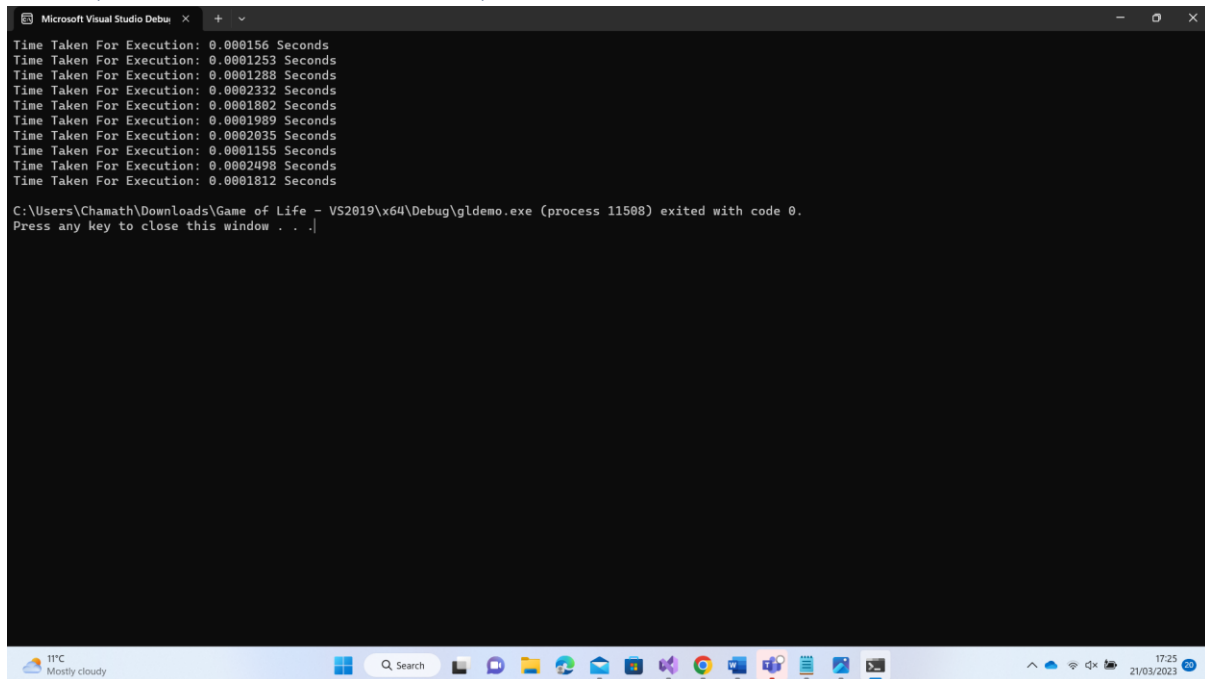


Figure 6

2.1– Plan 1 (Testing the Sequential Version with Pattern 1 -Dead)

For all the experiments, average times will be calculated and used as it would give an indication of the time required while a single timing result may be skewed due to higher CPU load at the time of the code was profiled.

Test 1 (Same Pattern, First Execution) :



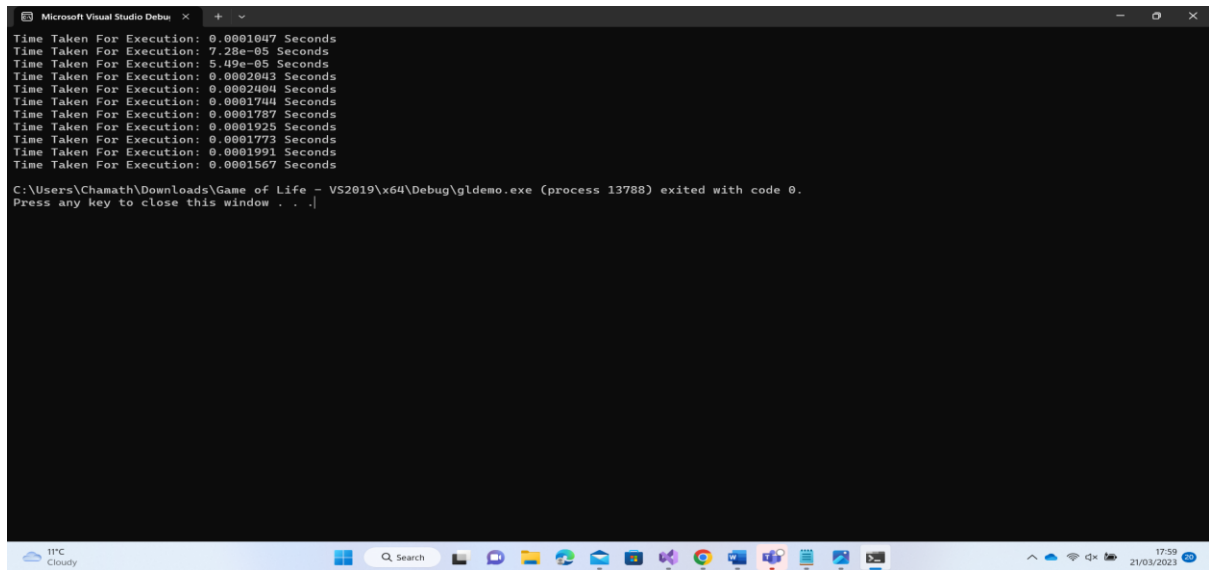
```
Microsoft Visual Studio Debug
Time Taken For Execution: 0.000156 Seconds
Time Taken For Execution: 0.0001253 Seconds
Time Taken For Execution: 0.0001288 Seconds
Time Taken For Execution: 0.0002332 Seconds
Time Taken For Execution: 0.0001802 Seconds
Time Taken For Execution: 0.0001989 Seconds
Time Taken For Execution: 0.0002035 Seconds
Time Taken For Execution: 0.0001155 Seconds
Time Taken For Execution: 0.0002498 Seconds
Time Taken For Execution: 0.0001812 Seconds
C:\Users\Chamath\Downloads\Game of Life - VS2019\x64\Debug\gldemo.exe (process 11508) exited with code 0.
Press any key to close this window . . .|
```

Figure 7

Average time taken = $0.0015922/10$

Therefore, the average time until that is 0.00015922 seconds.

Test 2 (Same Pattern, Second Execution) :



```
Microsoft Visual Studio Debug
Time Taken For Execution: 0.0001047 Seconds
Time Taken For Execution: 7.28e-05 Seconds
Time Taken For Execution: 5.49e-05 Seconds
Time Taken For Execution: 0.0002042 Seconds
Time Taken For Execution: 0.0002404 Seconds
Time Taken For Execution: 0.0001744 Seconds
Time Taken For Execution: 0.0001787 Seconds
Time Taken For Execution: 0.0001925 Seconds
Time Taken For Execution: 0.0001772 Seconds
Time Taken For Execution: 0.0001991 Seconds
Time Taken For Execution: 0.0001567 Seconds

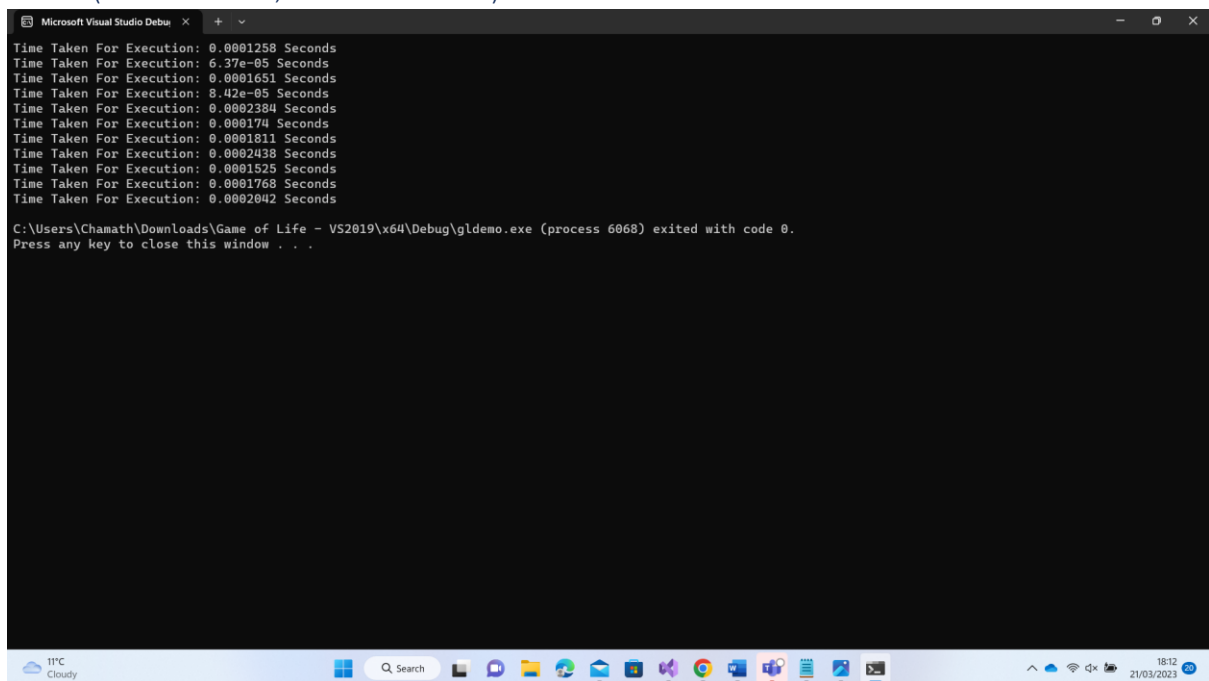
C:\Users\Chamath\Downloads\Game of Life - VS2019\x64\Debug\gldemo.exe (process 13788) exited with code 0.
Press any key to close this window . . .
```

Figure 8

Average time taken = $0.00016954 / 10$

Therefore, the average time until that is 0.00016954 seconds.

Test 3 (Same Pattern, Third Execution) :



```
Microsoft Visual Studio Debug
Time Taken For Execution: 0.0001258 Seconds
Time Taken For Execution: 6.37e-05 Seconds
Time Taken For Execution: 0.0001651 Seconds
Time Taken For Execution: 8.42e-05 Seconds
Time Taken For Execution: 0.0002384 Seconds
Time Taken For Execution: 0.000174 Seconds
Time Taken For Execution: 0.0001811 Seconds
Time Taken For Execution: 0.0002438 Seconds
Time Taken For Execution: 0.0001525 Seconds
Time Taken For Execution: 0.0001768 Seconds
Time Taken For Execution: 0.0002042 Seconds

C:\Users\Chamath\Downloads\Game of Life - VS2019\x64\Debug\gldemo.exe (process 6068) exited with code 0.
Press any key to close this window . . .
```

Figure 9

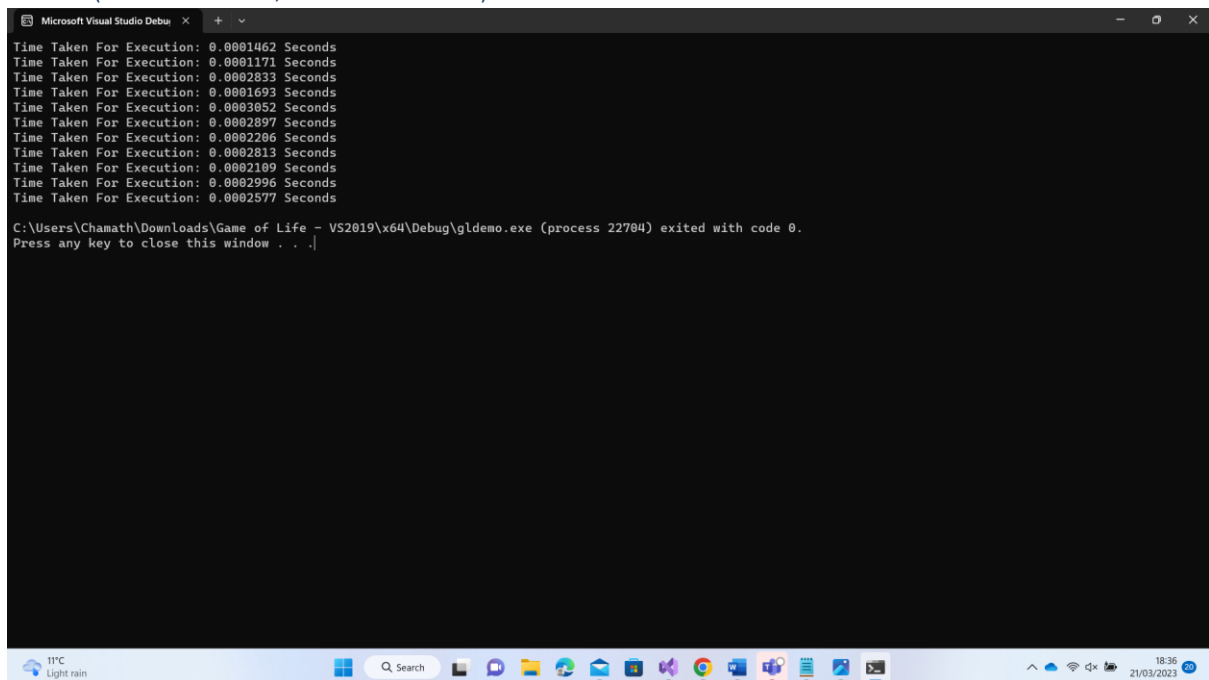
Average time taken = $0.00016356 / 10$

Therefore, the average time until that is 0.00016356 seconds.

Here, three tests have been conducted for Sequential Version and found three average times taken for each testing's execution. All the averages times in the above three tests are slightly different. Therefore, now I can come to a conclusion about average time the Sequential Version takes. Thus, the average time taking for execution inside the Sequential Version will approximately be 0.00016477 Seconds.

2.2 – Plan 2 (Testing the Parallel Version with Pattern 1 -Dead)

Test 1 (Same Pattern, First Execution) :



```
Microsoft Visual Studio Debug Console
Time Taken For Execution: 0.0001462 Seconds
Time Taken For Execution: 0.0001171 Seconds
Time Taken For Execution: 0.0002833 Seconds
Time Taken For Execution: 0.0001693 Seconds
Time Taken For Execution: 0.0003852 Seconds
Time Taken For Execution: 0.0002897 Seconds
Time Taken For Execution: 0.0002206 Seconds
Time Taken For Execution: 0.0002813 Seconds
Time Taken For Execution: 0.0002109 Seconds
Time Taken For Execution: 0.0002996 Seconds
Time Taken For Execution: 0.0002577 Seconds

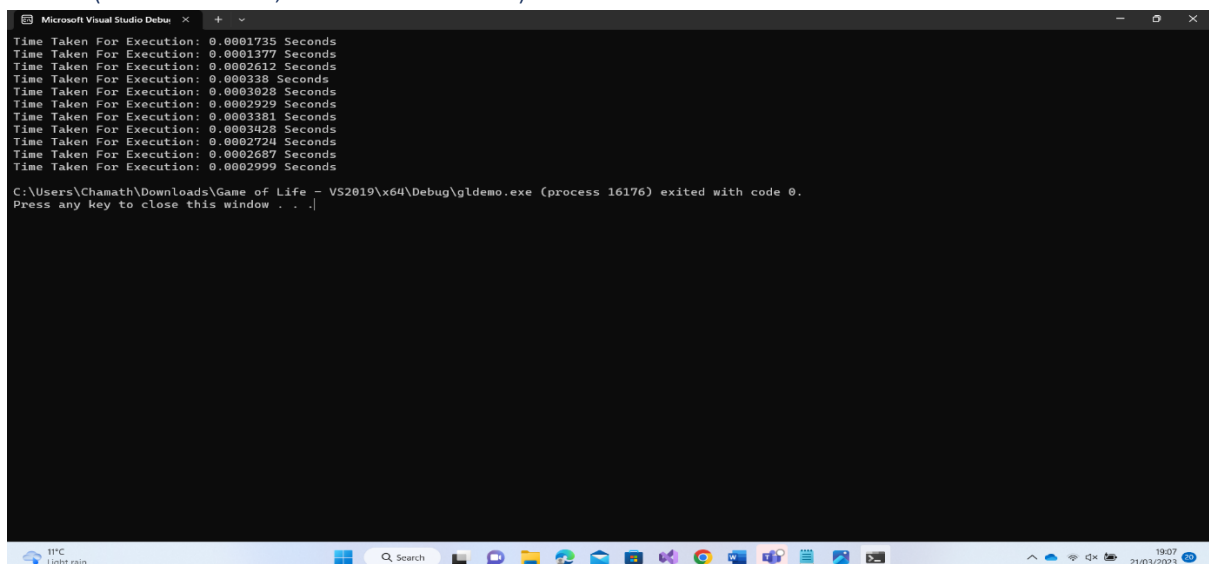
C:\Users\Chamath\Downloads\Game of Life - VS2019\x64\Debug\gldemo.exe (process 22704) exited with code 0.
Press any key to close this window . . .|
```

Figure 10

Average time taken = $0.0025130/10$

Therefore, the average time until that is 0.0002513 seconds.

Test 2 (Same Pattern, Second Execution) :



```
Microsoft Visual Studio Debug Console
Time Taken For Execution: 0.0001735 Seconds
Time Taken For Execution: 0.0001377 Seconds
Time Taken For Execution: 0.0002612 Seconds
Time Taken For Execution: 0.000338 Seconds
Time Taken For Execution: 0.0003828 Seconds
Time Taken For Execution: 0.0002929 Seconds
Time Taken For Execution: 0.0003381 Seconds
Time Taken For Execution: 0.0003428 Seconds
Time Taken For Execution: 0.0002724 Seconds
Time Taken For Execution: 0.0002687 Seconds
Time Taken For Execution: 0.0002999 Seconds

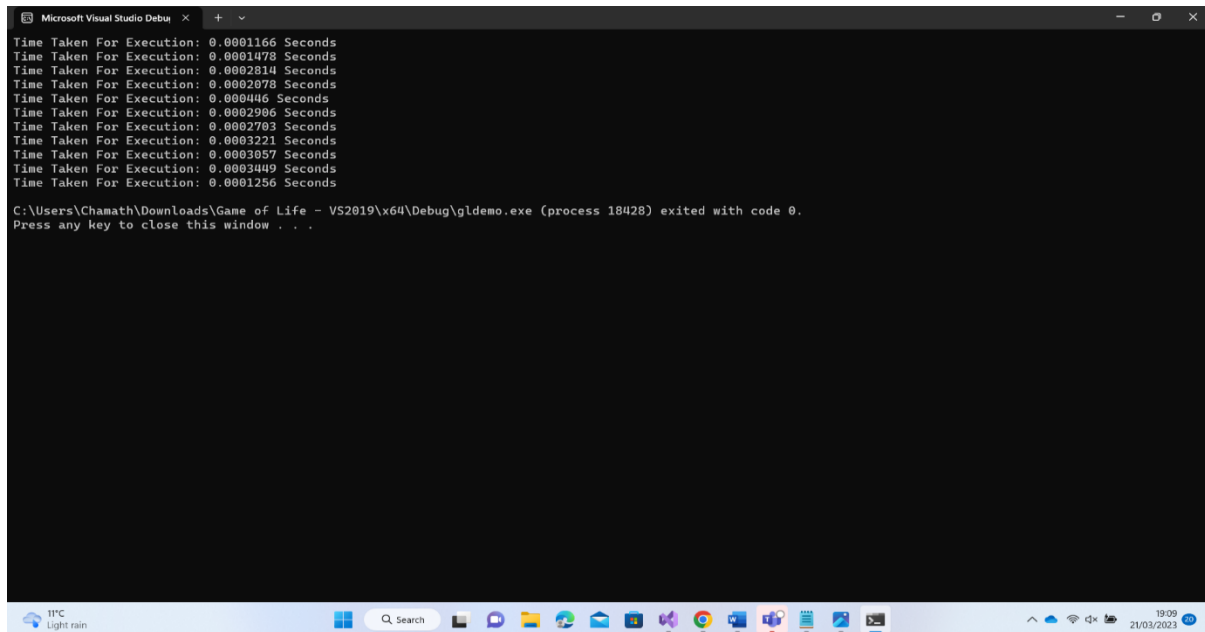
C:\Users\Chamath\Downloads\Game of Life - VS2019\x64\Debug\gldemo.exe (process 16176) exited with code 0.
Press any key to close this window . . .|
```

Figure 11

Average time taken = $0.0028054/ 10$

Therefore, the average time until that is 0.00028054 seconds.

Test 3 (Same Pattern, Third Execution) :



```
Microsoft Visual Studio Debug
Time Taken For Execution: 0.0001166 Seconds
Time Taken For Execution: 0.0001478 Seconds
Time Taken For Execution: 0.0002814 Seconds
Time Taken For Execution: 0.0002078 Seconds
Time Taken For Execution: 0.000446 Seconds
Time Taken For Execution: 0.0002906 Seconds
Time Taken For Execution: 0.0002703 Seconds
Time Taken For Execution: 0.0003221 Seconds
Time Taken For Execution: 0.0003857 Seconds
Time Taken For Execution: 0.0003449 Seconds
C:\Users\Chamath\Downloads\Game of Life - VS2019\x64\Debug\gldemo.exe (process 18428) exited with code 0.
Press any key to close this window . . .
```

Figure 12

Average time taken = $0.00028332 / 10$

Therefore, the average time until that is 0.00028332 seconds.

As the pattern used is a Dead Pattern, I have manually stopped the execution when the graphical presentation of the pattern is ended during each simulation. Thus, the above attached screenshots depict the times taken for the execution until its manually stopped. For all six tests in the experiment, a fixed array size of 30×30 is employed. Here, three tests have been conducted for Parallel Version and noticed one thing. That is, from the three findings that were obtained, the first test's average time is 0.0002513 seconds and the second and third tests' average times are, respectively, 0.00028054 and 0.00028054 seconds. What is observable is the average times in both tests two and three are slightly different whereas average time in Test one has a significant difference between the average times consumed in Test 2 and Test3. That's called a lower Latency.

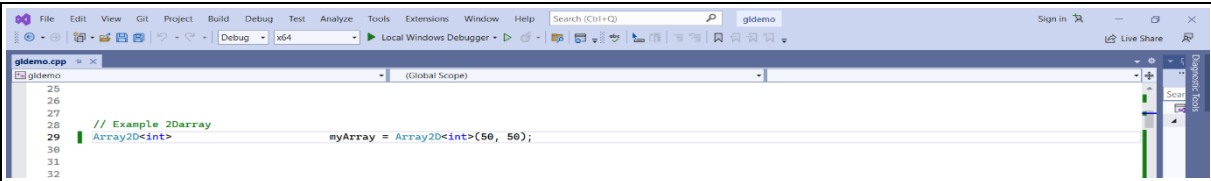
Comparison

Therefore, can come to a conclusion about average times. This is due to the pre-emptive multi-tasking environment in which the programme operates, in which the OS manages process allocation time on the CPU. **As a result, from one run time of the programme to the next, the CPU time allocation for the programme (process) has been varied.** Test 1 in the Parallel Version becomes considerable because it depicts the lowest average time among six tests in both versions. That's because, at the processing time of test 1 in the Parallel Version, it "simultaneously uses multiple processing cores" (Foundation, 2016–2020) from the CPU. But during the execution process of Test 2 and Test 3, the CPU has failed to provide a faster and same processing power by allocating time for the programme. When it comes to the average times in the tests of Sequential Version, it is apparent that "Sequential operators execute on a single processing node" (Corporation, 2007, 2019) and that's why the average times are not significantly different and slightly differed in each test. Thus, this experiment 1 is crucial because it demonstrates that the process allocation time by the OS on the CPU affects the performance of program.

3 – Experiment 2 (Varying the Array Size)

In the below table representations, it shows three Plans in which the Array Size is different in each plan. The first plan is tested using the array size of 50 x 50, the second plan is tested using the array size of 100 x 100 and then gradually increased the array into 1000 x 1000. **Each plan includes one Dead Pattern and one Repeat Pattern and have compared the Average Time with both the Sequential and Parallel Versions.** In order to obtain the Average Time, **two tests have been conducted inside every plan and to calculate the average time taken for each test**, manually stopped the execution of the pattern when ten executions completed.

3.1 - Plan1 (Array Size = 50 x 50)



	Sequential	Average Time (Seconds)	Parallel	Average Time (Seconds)
Dead Pattern 1	Test 1	0.00025029	Test1	0.00031957
	Test 2	0.00029654	Test2	0.00031165
	Total Average	0.00027341	Total Average	0.00031561
Repeat Pattern 3	Test 1	0.00024734	Test1	0.00024014
	Test 2	0.00023332	Test2	0.0002047
	Total Average	0.00024033	Total Average	0.00022242

Table 1

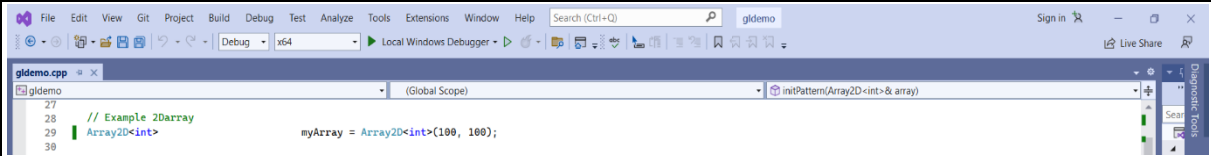
In the above experiment, the array size is fixed to 50 x 50. When executing the Sequential Version using Dead Pattern 1, the average run time received after the calculation is **0.00027341** seconds whereas the average run time received for the Parallel Version is **0.00031561** seconds which outlines that the Parallel Version has taken more time for the simulation with a difference of 0.0000422 seconds. So, can come to a conclusion as even though Parallel Version has taken more time than the Sequential Version, there is only a slight difference between the run times which highlights that the Parallel Version's performance has had a higher latency as the CPU time allocation for the programme (process) has been varied due to low processing power in the CPU or due to unavailability of multiple processing cores to be utilized.

When executing the Sequential Version using Repeat Pattern 3, the average run time received after the calculation is **0.00024033** seconds whereas the average run time received for the Parallel Version is **0.00022242** seconds which is a significant point because the Parallel Version has taken less time for the simulation when compared with the average time taken in Sequential Version. Another important observation is the very less amount of time taken for

the Test 2 in the Parallel Version which is recorded as 0.0002047 seconds. When it's compared with the Test 1 in the Parallel Version, its identical that Test 2 has had a faster stimulation.

Therefore, it's possible to state that **Parallel Version is not the fastest all the time and the speedup or the performance is affected and depends on the CPU time allocation provided by the Operating System.**

3.2 - Plan2 (Array Size = 100 x 100)

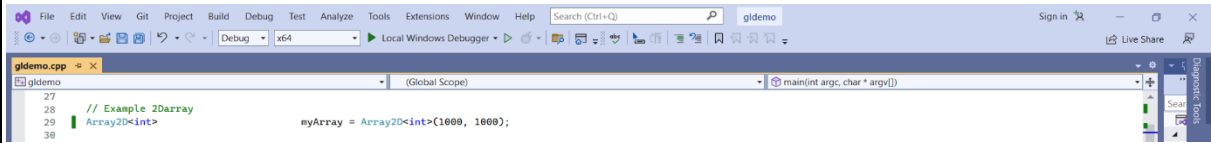


	Sequential	Average Time (Seconds)	Parallel	Average Time (Seconds)
Dead Pattern 2	Test 1	0.00074182	Test1	0.00038915
	Test 2	0.00072896	Test2	0.00037568
	Total Average	0.00073539	Total Average	0.00038241
Repeat Pattern 4	Test 1	0.00071473	Test1	0.00039436
	Test 2	0.00071426	Test2	0.00038743
	Total Average	0.00071450	Total Average	0.00039089

Table 2

In the above experiment, the array size is fixed to 100 x 100. This experiment's outcomes are more straight forward than the previous experiment. When executing the Sequential Version using Dead Pattern 2, the average run time received after the calculation is **0.00073539** seconds whereas the average run time received for the Parallel Version is **0.000382415** seconds. When executing the Sequential Version using Repeat Pattern 4, the average run time received after the calculation is **0.00071450** seconds whereas the average run time received for the Parallel Version is **0.000390895** seconds. With those values, its apparent that the both the patterns in the Sequential Version have only an insignificant time variation whereas it's the same for both the patterns in the Parallel Version. But **there can be seen a dramatic change during the average run times between the Sequence and Parallel Versions in both the patterns.** That's, the average times taken for both the Patterns in the Parallel Version are nearly a half of the average times taken for the same Patterns in the Sequential Version. That means, there had been a faster simulation from the Parallel Version during this experiment phase. **Thus, this speedup obtained from the Parallel Version provide evidence that the assumption that the Parallel Version will execute faster can occasionally be satisfied.**

3.3 - Plan3 (Array Size = 1000 x 1000)



The screenshot shows a C++ IDE with a file named 'gldemo.cpp'. The code defines a 2D array 'myArray' of size 1000x1000. Below the code is a table with 5 columns: Sequential, Average Time (Seconds), Parallel, and Average Time (Seconds). The table contains data for two patterns: 'Dead Pattern 2' and 'Repeat Pattern 4'. For 'Dead Pattern 2', the Sequential average time is 0.079431115 seconds and the Parallel average time is 0.01990685 seconds. For 'Repeat Pattern 4', the Sequential average time is 0.076110021 seconds and the Parallel average time is 0.02196835 seconds.

	Sequential	Average Time (Seconds)	Parallel	Average Time (Seconds)
Dead Pattern 2	Test 1	0.07984328	Test1	0.01990774
	Test 2	0.07901895	Test2	0.01990596
	Total Average	0.079431115	Total Average	0.01990685
Repeat Pattern 4	Test 1	0.07754177	Test1	0.0207903
	Test 2	0.07467827	Test2	0.0231464
	Total Average	0.076110021	Total Average	0.02196835

Table 3

In the above experiment, the array size is fixed to 1000 x 1000. In this experiment also, the outcomes are more straight forward as the previous experiment. When executing the Sequential Version using Dead Pattern 2, the average run time received after the calculation is **0.079431115** seconds whereas the average run time received for the Parallel Version is **0.01990685** seconds. When executing the Sequential Version using Repeat Pattern 4, the average run time received after the calculation is **0.076110021** seconds whereas the average run time received for the Parallel Version is **0.02196835** seconds. With those values, it's apparent that the both the patterns in the Sequential Version have only a slight time difference whereas it's the same in the Parallel Version. However, there's a considerable change during the average execution times between the Sequence and Parallel Versions in both the patterns. That's, the average times taken for both the Patterns in the Sequence Version are approximately three times of the average times taken for the same Patterns in the Parallel Version. That means, there had been an excellent and faster simulation from the Parallel Version during this time. Therefore, it's convenient to mention in here also that the speedup obtained from the Parallel Version prove the fact that the Parallel Version may enable to simulate the same task multiple times faster.

Comparison

In addition to the time differences and speed, it is noticeable that when the array size increases, the time taken to execute also has been increased significantly. Furthermore, the same patterns are used for testing and there cannot identify a big difference between the average times taken. But the only factor affects the performance is the array size as it increases time taken to simulate when the array size is increased. Then the speed will be reduced and it directly affects the performance. Thus, the above Experiment proves that “a parallel program works faster than its serial version used to solve the same problem”. But also “Many conflicting parameters such as parallel overhead, hardware architecture, programming paradigm, programming style may negatively affect the execution time of a parallel program making its execution time larger than that of the serial version and thus any parallelization gain will be lost”. Therefore, “to obtain a faster parallel program, these conflicted parameters need to be well optimized” too. Finally, its proven that this experiment is crucial because it enables to observe how the algorithm expands as the size of the array increases. (El-Nashar, 2011)

4 – Analysis Made from Experiment Outcomes

Finally, its proven how the various settings of each experiment affect performance by comparing the run-times. In most of the cases, the testing depict that the Parallel Version is faster than the Sequential Version as a parallelized algorithm has the ability to share the workload among several cores. Therefore, it is apparent that parallelization can aid in achieving significantly better performance by enabling task acceleration. (Kung, 1976)

But in some circumstances, the performance resulted relatively low in Parallel Version when contrasted with the Sequential Version. One of the key aspects can be the time allocation by the CPU for a particular program from the OS to run, computing capability of the device used and the availability of many CPU cores. So, the hardware occasionally limits the advantages of parallel processing. Nevertheless, a parallel program's simulation time may be longer than that of the serial version due to a number of other constraints, such as parallel overhead and programming paradigm, which will lessen any gains from parallelization. (El-Nashar, 2011) Therefore, it's apparent that performance measures the latency, throughput, power consumption and computational work done in completing a given task.

Conclusion

In conclusion, the two implementations Sequential and Parallel versions of Conway's Game of Life which use the C++ and TBB libraries have been implemented successfully. The implementation of the rules also has been completed. In this paper, many number of experiments have been conducted and the results were displayed along with evidence. Those experiments and the descriptions aid in providing a clear understanding on how the two kinds of implementations executes in terms of the version, a fixed array size and different array/universe sizes. The outcomes of those experiments pave how various properties affect the performance of the implementation. Overall, this been a great opportunity to study about parallel programming concepts into practice.

References

Corporation, I., 2007, 2019. *Defining parallel and sequential operators*. [Online] Available at: <https://www.ibm.com/docs/en/iis/11.7?topic=operators-defining-parallel-sequential> [Accessed 11 03 2023].

El-Nashar, A. I., 2011. TO PARALLELIZE OR NOT TO PARALLELIZE, SPEED UP ISSUE. *International Journal of Distributed and Parallel Systems*, Vol.2(No.2), p. 15.

Foundation, S. C., 2016–2020. *Introduction to Parallel Computing using MATLAB*. [Online] Available at: <https://researchcomputingservices.github.io/parallel-computing/01-parallel-introduction/#:~:text=In%20contrast%2C%20with%20parallel%20computing,running%20an%20equivalent%20sequential%20program> [Accessed 11 03 2023].

Kung, H. T., 1976. SYNCHRONIZED AND ASYNCHRONOUS PARALLEL ALGORITHMS. *SYNCHRONIZED AND ASYNCHRONOUS PARALLEL ALGORITHMS*, Vol 41(Part II), p. 50.

LINK to the Project Folder:

https://outlookuwicac.sharepoint.com/:f:/s/CIS6007_T2_22ParallelandDistributedSystems_T2_22/EpfC0BjMFg9GmoKMgITBLAUByBkgdZTSDnNOd-rvMo8Tbg?e=YCL3Gw

LINK to the Recording:

https://outlookuwicac.sharepoint.com/:f:/s/CIS6007_T2_22ParallelandDistributedSystems_T2_22/EuXf3nyXAk9KtBKZYYfzADcBWXalmo5M98NUffsmjJSwHw?e=skZYkw