2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE) 978-1-7281-4142-8/20/$31.00 ©2020 IEEE 10.1109/ic-ETITE47903.2020.107

Performance Enhancement Of Conway's Game Of Life Using Parallel And Distributed Computing Platform

Rhythm Goyal
School of Computer Science and Engineering
Vellore Institute of Technology, Vellore
rhythmgoyal98@gmail.com

Charan Lalchand Soneji
School of Computer Science and Engineering
Vellore Institute of Technology, Vellore
charan.soneji.cls@gmail.com

Bhanu Venkata Kiran Velpula
School of Computer Science and Engineering
Vellore Institute of Technology, Vellore
beekiran00@gmail.com

Sairabanu J.
Associate Professor
Vellore Institute of Technology, Vellore
jsairabanu@vit.ac.in

***Abstract*- The game of life, a cellular automation model, is based on the principles of emergence and self-organization. It is a zero-player game that plays out on the basis of its initial state only with the help of some simple rules. It is an infinite computational process which may generate complex patterns over time. In this paper, we propose and compare 2 different methods along with their architectures to optimize the functioning of the existing method. The first method uses OpenMP which works on the fundamentals of shared memory architecture whereas the second method uses MPI which in turn uses message passing techniques on distributed memory systems.**

Keywords- Conway's Game of Life, OpenMP, MPI, Serial, Game Programming, parallel, distributed, Amdahl, performance

I. INTRODUCTION

Conway's game of life is an application of Cellular automation which is described within this paper. Conway's game is used to describe cellular automata and its expansion in many corresponding iterations without having two bother about the calculations of each and every cell defined within the grid. Cellular automation is something that is something whose application is used in a wide variety of fields such as fluid and lava flow, viral infection dynamics, social computational systems and is also used in understanding the behavior of cells along a given fixed set of conditions. Supposing we start with a given network of cells in the first iteration, Conway's game of life carries out all the required computations and displays the pattern in the next iteration based on those specific set of rules which are defined.

Within this paper, we shall compare the serial and parallel execution of the code for Conway's game of life which is run on different parallel platforms and see how they are useful for cellular automation and its several iterations and discuss the variation in efficiency of the code when it is run serially and parallelly on the same CPU. However, we shall not compare different models of cellular automata as that was not the purpose of the paper.

A. Contributions

Parallel computing helps in making efficient use of the given PC and all of its cores and reducing the overall computation time and it is also used for producing a better speedup as output and PC's with 16 or 32 cores and a decent amount of RAM are widely used for such applications.
The contributions of this paper are mentioned below:

- We have discussed two different parallel platforms in order to run Conway's game of life in a parallel manner.
- We have discussed and compared the results obtained and contrasted serial results with parallel outputs and have discussed about the speedup as well.
- We have concluded using a brief explanation of the hardware and software details.

B. Paper Overview

In the serial section of this paper, it has been discussed how the best performance and output can be obtained on a single core and the quickest time taken for all the computations on a single core whereas on the parallel section of the paper, we have discussed about 2 different platforms along which the parallel code is run and the difference among them and how the speedup gets affected. In the section after that, we have compared the serial and parallel outputs and discussed about the overall system performance. To run the parallel codes on their respective platforms, we have used the following hardware:

TABLE I.

System	Lenovo Legion Y540
Number of Processors	1
Processor model	Intel Core i7-9750H
Launch year	02 2019
Cores per processor	6
Bits per word	64
Frequency	2.60 - 4.50 GHz

Main Memory	128GB DDR4-2666

Fig. 1. Evaluation System

In this paper, Section II represents the Literature Survey, Section III represents the basics of Game of Life, Section IV talks about the existing serial system of executing the code, Section V mentions the first proposed parallelized system OpenMP, Section VI consists of the second proposed parallelizing system MPI, Section VII refers to the Comparison between both the proposed systems and finally Section VIII mentions the experimental results obtained after the completion of the project.

II. LITERATURE SURVEY

- Parallel Computing Models and Analysis of OpenMP Optimization on Intel i7 and Xeon Processors[1] – This paper reviewed the hardware and software architectures of both the devices and APIs like OpenMP ,OpenCL and CUDA. It compared the performances by optimizing an image processing code on clustering using remote sensing data. The comparison of Open MP, CUDA, OpenCL and OpenACC parallel programming techniques and the research on individual benefits and limitations was very advantageous. The code optimization of k-mean clustering using OpenMP 2.0 along with explanations and observation data using graphs, tables and flowcharts helped in clear understanding. But no future enhancements were proposed.
- The Use of Game Development in Computer Science and Software Engineering Education[2] – This paper is a literature survey about research articles on game development from 2004-2012. It concludes by talking about the best tools and game development frameworks for teaching Software Engineering and Computer Science. It provides recommendations of useful frameworks, provides answers to 3 major questions about game development in software architecture and uses tables and charts to show supportive data.
- Design and Improvement of the Parallel Algorithm in the Computer Games System[3] - It compared the parallel and mononuclear search algorithms. Some improvements were added to the parallel methods and compared with the original. The Alpha-Beta implementation programming skills of the parallel search methods were discussed. The paper doesn't take about future scope.
- Game Design Research[4] – This paper tried to attract attention towards the lack of focus on game study, game design and its research. It suggested that the frameworks of design research should be improved as it in turn improves our understanding of the design of the game. It primarily explains the difference between game study, game design and design research. This paper sets a background and encourages future research in the field of game studies and aims on bringing topics of game studies, game design closer.
- Parallel Computing using OpenMP[5] - This paper talks about using APIs like OpenMP to improve the speedup and eventually the system performance by explaining the basic concept of parallel programming, i.e., splitting of a program by multiple processors simultaneously, leading to higher performance. It gives insight on the disadvantages using OpenMP and the applications of OpenMP in real-world. No future enhancements were looked into.
- Exploring parallel game architectures with tardiness policy[6] - They proposed a parallel and adaptive architecture which uses the tardiness policy which is done to handle issues like wastage of computer resources. It is used to monitor and change task behavior based on the then conditions of the host hardware. 2 examples are used to show the working of the architecture. The upper hand was that they applied tardiness control to adapt task operation and developed 2 prototypes using proposed architecture. Future enhancement that is possible is to create scheduling strategies to maximize game task processing with multicore CPUs and programmable GPUs.
- Impact of Thread Synchronization and data Parallelism on Multicore Game Programming[7] - They research on the challenges and benefits of thread synchronization and data level parallelism on multicore game engine programming. They implement an interactive game in an 8-core system using 4 different models (STM, MAM, MSM, MSMDP). The results of all 4 are compared based on the execution time. The positive in this paper was that they introduced the fact that new multicore systems have the potential to improve the performance of multithreaded game engines, observation data, graphs. The disadvantage was that a single processor multithreaded game engine struggles to improve performance due to lack of hardware support. The system can be further improved by implementing the entire test game engine on a CPU/GPU platform to explore the performance and power in the next project.
- Parallelization : Conway's Game of Life[8] - Talks about the Game of Life and how we can formulate an approach to parallelize the game. It covers 3 versions – shared memory, MPI and a hybrid. It guides us how to measure the performance and scaling of a parallel application in multicore and manycore applications. An advantage insight on cellular automation and uses in scientific problems, measure of performance and scaling of parallel applications, algorithms and implementation shown. But no future enhancements were suggested.
- A comparison of Parallel Design Patterns for Game Development[9] – This paper provides guidelines as to when to use which of the 2 parallel design patterns- fork-join or pipeline parallelism. A data set was used to compare the two on the basis of speedup and efficiency. Both patterns were applied

on a variety of test cases for 2 different game branch loops- BOIDS and Animation system. The advantage was the implementation of two parallel design patterns, their comparison in speedup and efficiency, flowcharts and graphical representations of observations and important algorithms were also discussed. Future enhancements proposed was to understand how parallelism can be utilized for performance optimization in games and testing of more patterns.

- Parallelizing a real time Steering Simulation for computer games with Open MP[10] – In this paper, they worked on parallelizing the C++ application similar to games – Open Steer Demo with OpenMP which they achieved by changing the high level design and refactored the interfaces to achieve explicit shared resource access. The advantage in this paper was that, problems for parallelization were discussed and the performance was analyzed between original and parallel versions along with graphical representations of observations. The disadvantage we found was the algorithms or flowcharts were not discussed for parallelization. Also, no future enhancements were proposed.

III. GAME OF LIFE

Before The Conway's Game of Life is a computational two-dimensional grid in which each cell is in either a living or a dead state. In this game, the player only interacts with it in the initial stages. From there the cell reproduction is an infinite and automatic process. The destiny of a cell is identified by 4 simple rules which are based on the states of its 8 adjacent cells. The rules are as follows:

- Birth: Each dead cell adjacent to exactly three live neighbors will become live in the next generation.
- Death by isolation: Each live cell with one or fewer live neighbors will die in the next generation.
- Death by overcrowding: Each live cell with four or more live neighbors will die in the next generation.
- Survival: Each live cell with either two or three live neighbors will remain alive for the next generation

These computations may sometimes lead to very complex patterns.

As the game of life is an example of cellular automata, it is very useful for modelling complicated non-linear systems such as in physics, chemistry, biology, meteorology, cosmology, computational science, etc. One of its most notable uses is when Stephen Hawking explained the working of the universe using game of life as an example and the creation and understanding on the Turing Machine as well.

IV. EXISTING SYSTEM: SERIAL METHOD

Conway's game of life deals with the development of patterns in each iteration based on 4 sets of rules. Now the number of calculations in each iteration may not be very high but as we keep proceeding, the speed or the time taken to for each of the iteration starts increasing and it is very high as compared to the time taken if the calculations were to be done in a parallel environment. The entire motive of serial code is to execute everything in a step-wise manner as it may be seen in Fig. 2. There is no specific library used for serial execution of code because each of the cores run it in a serial manner by default without considering the number of cores available and even if it could be run in a more efficient manner.

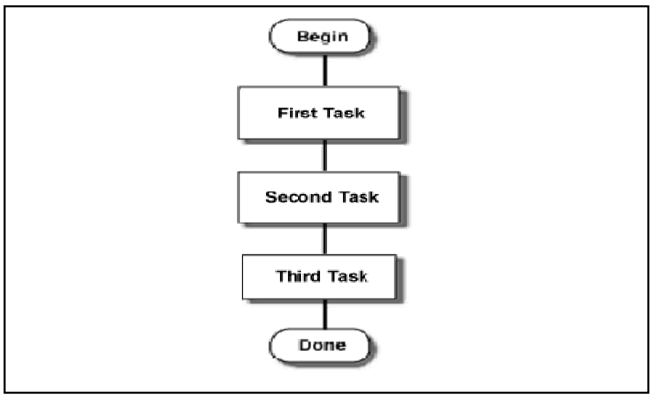

Fig. 2. Basic workflow of serial execution

In a serial environment, all the calculations are to be done by a single core of the CPU rather than making use of all the cores of the CPU. The main purpose of doing serial execution is to compare the time taken and the speedup of parallel execution along with serial execution. The apparent speed up could be wrong at times and hence it is better to obtain a speedup from a slow baseline since the overhead would be small as relatively compared to the execution times. We will call superfluous presumptions getting from the supposition of sequential execution sequential snares. From quite a while ago supported sequential figment has made various sequential snares gotten incorporated with both our apparatuses and perspectives. A significant number of these power serialization due to over-particular of the calculation. It isn't so much that software engineers needed to compel serialization; it was basically accepted. Since it was advantageous and there was no punishment at the time, sequential suspicions have been consolidated into about everything.

A. *Representations*

Cell representations basically tell us whether a cell is alive or dead and in order to save memory, there is a method that is used whereby an alive cell is given a "cell bit" of 1 and the dead state of a cell is represented by a "cell bit" of 0. For the calculation process, it is not required that each of these bit state values be extracted. Now supposing we take a 3x3 matrix, and consider the middlemost element to be the cell. Then the pattern of the elements around it would need to be calculated. Hence for each cell, a total of 8 calculations needs to be made. Based on the CPU and its registers, the bit values shall be stored.

B. *Avoiding Unnecessary computations*

In order to avoid computations which can indirectly reduce the load on the CPU based on the calculations to be made, the areas of each iteration which remain stagnant for more than 2 iterations can be left as it is and be loaded directly from the register storing the values rather than computing the values for them. This way, the amount of work to be done by the CPU gets reduced to a very large extent.

C. *Working*

The flowchart given below explain about the execution of the serial code for Conway's game of life. The game has several different functions which are all used for each iteration and each core processes each of the functions one at a time in a sequential manner. The alive cell function reads and checks if the cell state is '0' or '1' which defines if it is alive or dead and then based on whether it is alive or not, it moves onto the Set borders which duplicates the row from the previous iteration and interchanges the corner elements as a rule of Conway's game of life. From here, when it comes to analyze cell function, the rules of the game are applied and the next state of the cell bit is calculated and the 4 different values are obtained based on the four different rules that are used in Conway's game of life.

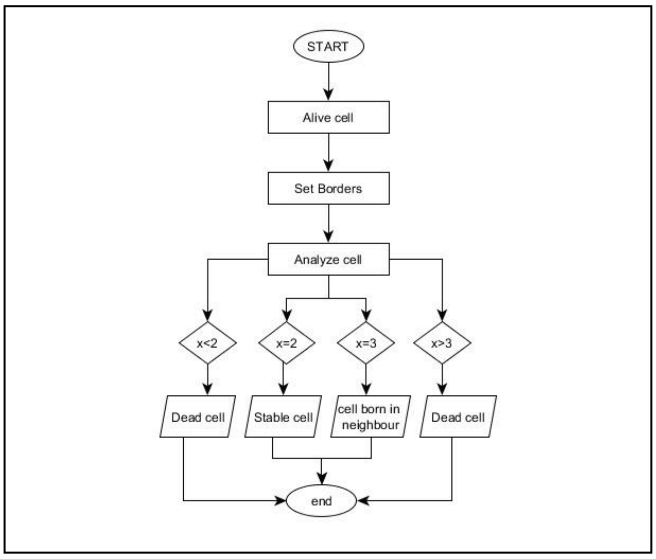

Fig. 3. Serialized Execution

V. PROPOSED SYSTEM 1: USING OPENMP

In this section, we propose a system using OpenMP which is a library used for parallel programming in shared memory processors models. C++, C and Fortran are supported by this library. The code is divided into sections- the sequential section (responsible to initialize variables, set up the programs' environment) and the parallelized (consists of the code that has to be divided amongst cores for faster and better execution) section.

One thread that runs from start to the end is known as the master thread whereas the rest of them are known as slave threads. All the threads in OpenMP share both memory as well as the data. A derivative (OMP pragma) marks the beginning of the code in the parallel section. Here the main thread forks into several slave threads that run the parallel part of the code. Once it is done, all these threads join back to form the master thread. We have used this library because it is very efficient, allows low level parallel codes. For easy understanding, it hides the low-level details from the programmer and uses only high-level constructs to describe the parallel code.

We have used OpenMP because it allows the programmer to specify the parallel section in the code, tell if the variables used in the parallel part are shared or private, whether and how the threads are synchronized, explain the method to parallelize loops, and how scheduling is done among threads.

For this system, we have parallelized the main functions among the number of cores present in the system, which over here is 6. The master thread does the work till setting up the borders for the grid. After this, all the functions from analyze cell onwards gets split up into all the cores in the system which results in a drastic reduction in the time taken for the complete execution.

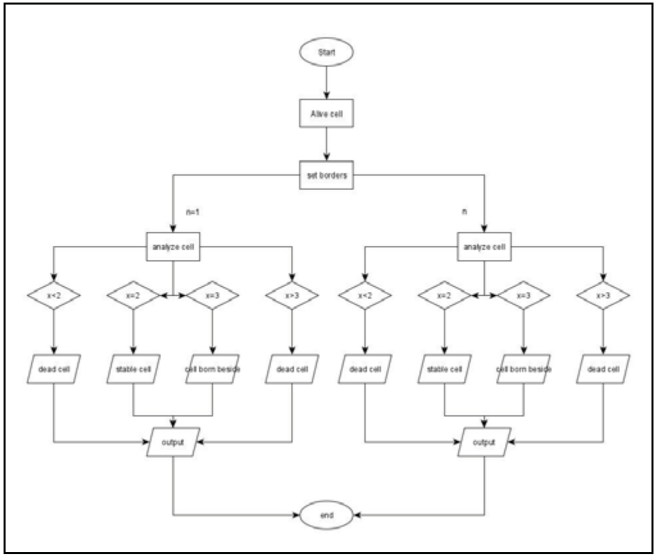

Fig. 4. OpenMP parallelized execution

Here we are comparing the results from the serial execution of the code with our first parallelization method – OpenMP. It is represented with the help of a graph for better understanding:

TABLE II.

Iterations	Serial Time(seconds) (T_s)	Parallel Time(seconds) (T_p)	Speedup T_s/T_p (seconds)
1	10.92	1.53	7.137
2	21.84	3.078	7.095
3	32.76	4.617	7.095
4	43.69	6.157	7.095
5	54.612	7.696	7.096
6	65.53	9.23	7.099
7	76.45	10.77	7.098

Fig. 5. Serial vs OpenMP

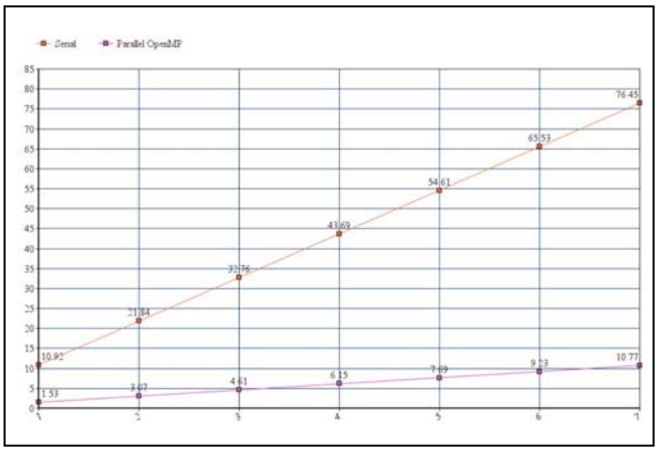

Fig. 6. Serial vs OpenMP graphical representation

Here, x-axis represents the number of iterations and the y-axis represents the time taken.

VI. PROPOSED SYSTEM 2: USING MPI

MPI or Message Passing Interface is a library used for message passing. The objective of this is to build up a convenient, proficient, and adaptable standard for message passing that will generally be utilized for composing message passing programs. All things considered, MPI is the principal institutionalized, seller free, message passing library.

The benefits of creating message passing programming utilizing MPI intently coordinate the structure objectives of convenience, effectiveness, and adaptability. It has undergone many versions of updates and the most recent one being MPI-3.X. It has been designed in a way that its interface specifications are declared for C and Fortran90. It uses communicators which are defined using MPI routines and may have a collection of users to define it. MPI has several different components that are used in order to define and differentiate the code.

MPI_INIT is used to initialize the MPI environment along with MPI_COMM_SIZE which is used to specify the number of processors or communicators to use.

In an MPI system, the program first begins with the serial code, followed by initializing its environment where we will have our parallel code. Once our environment is all set up, we begin executing the parallelized part. All the statements are executed along with message passing. This continues till the termination of the MPI environment where our parallel code ends. From there the serial code resumes.

The working our system will be explained in Fig. 7.

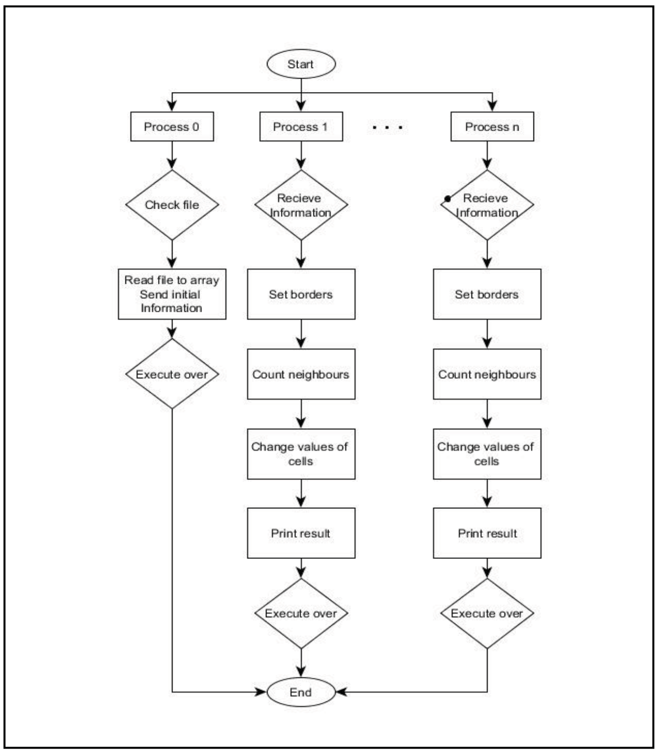

Fig. 7. MPI parallelized execution

Here we are comparing the results from the serial execution of the code with our second parallelization method – MPI. It is represented with the help of a graph for better understanding:

TABLE III.

Iterations	Serial Time(seconds) (T_s)	MPI Time(seconds) (T_p)	Speedup T_s/T_p (seconds)
1	10.92	0.596	18.322
2	21.84	1.192	18.322
3	32.76	1.788	18.322
4	43.69	2.384	18.326
5	54.612	2.98	18.326
6	65.53	3.576	18.324
7	76.45	4.172	18.324

Fig. 8. Serial vs MPI

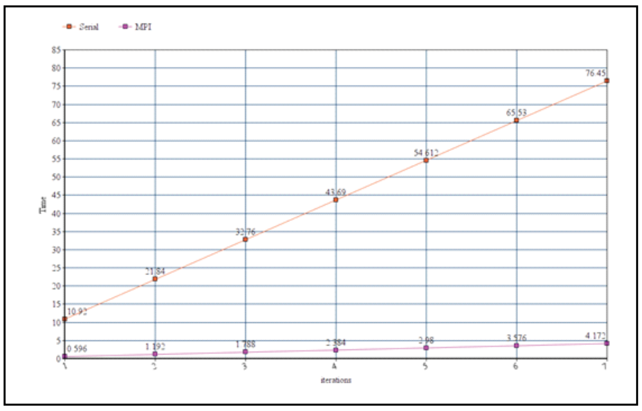

Fig. 9. Serial vs MPI graphical representation

Here, x-axis represents the number of iterations and the y-axis represents the time taken.

VII. COMPARISON

Serial processing takes a much longer time than parallel; In serial processing after adding the borders to the initial pattern the cell analysis is done for each cell dead or alive i.e., every cell's neighbor is being checked with the 4 conditions and the cell is changed accordingly. Meanwhile in parallel, the cell analysis is being spread across 6 different cores that the computer has, i.e., the work is being shared by multiple processors. In serial the data is transferred bit by bit whereas in parallel it is transferred in byte form. The number of processors play a major role in the reduction of workload.

It is clear from the Speed up times that parallel has more performance than serial processing and the time taken to extremely reduced for parallel processing.

In MPI parallelism occurs where every parallel process is working in its own memory space in isolation from others. All the processes in MPI are parallel and messages establish the dataflow structure of the computation. Clearly MPI method is faster compared to OpenMP parallel processing. Each process has their own local variables.

Usage of MPI requires more programming changes to go from serial to parallel version and the performance is limited by the communication network between the nodes.

Since OpenMP parallel processing is mostly used for loop parallelization, all the tests were done for the analyze cell function where every cell's neighbor is checked with the 4 conditions.

TABLE IV.

Iterations	OpenMP Time(seconds) (T_p)	MPI Time(seconds) (T_m)
1	1.53	0.596
2	3.078	1.192
3	4.617	1.788
4	6.157	2.384
5	7.696	2.98
6	9.23	3.576
7	10.77	4.172

Fig. 10. OpenMP vs MPI

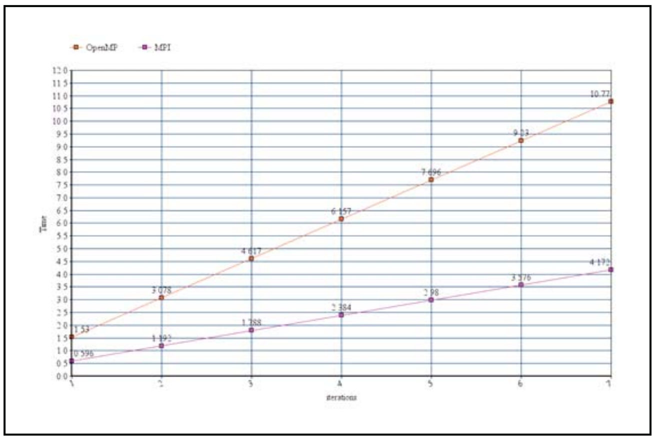

Fig. 11. OpenMP vs MPI graphical representation

This graph represents the comparison of time taken by the two parallelizing methods.

VIII. EXPERIMENTAL RESULTS

Amdahl's law states that, in parallelization, P being the proportion of the program that can be made parallel and 1-P being the proportion of the program that remains serial, the maximum speed up that can be achieved using N processors is

$$1/(1-P)+(P/N)$$

In other words, it is formula that gives the theoretical speed up in the execution of a task, by improving a particular part of the system.

Speed up: it is the ratio of performance by improving the task and performance by not improving the task.

Total speed up: Execution time without enhancing/execution time with enhancing.

We use Amdahl's law because we are running the application faster with the same problem size. Since our aim is to compare the difference in the processing times of Conway's game of life using serial, OpenMP and MPI. We are not using Gustafson's law because it gives the theoretical speed up of a task at fixed execution time with improved resources.

T_s : time taken for serial execution
T_p : time taken for parallel execution using OpenMP
T_m : time taken for parallel execution using MPI

Serial (1 iteration) $T_s = 10.92$s

OpenMP (parallel processing 1 iteration) $T_p = 1.53$s
Speedup S1 = $T_s/T_p = 10.92s/1.53s = 7.137s$

MPI (1 iteration) $T_m = 0.596$s
Speedup S2= $T_s/T_m = 10.92s/0.596s = 18.322s$

The total speed up for Serial and OpenMP parallel processing is 7.137s.

The total speed up for Serial and MPI parallel processing is 18.322s.

ACKNOWLEDGEMENT

We would like thank our teachers for providing us with the guidance and support that we needed.

REFERENCES

[1] Bays, C. (2010). Introduction to cellular automata and Conway's Game of Life. In Game of Life Cellular Automata (pp. 1-7). Springer, London.

[2] Bays, C. (1987). Candidates for the game of life in three dimensions. Complex Systems, 1(3), 373-400.

[3] Rendell, P. (2011, July). A universal turing machine in conway's game of life. In 2011 International Conference on High Performance Computing & Simulation (pp. 764-772). IEEE.

[4] Hua, D., & Pelikan, M. (2012). Variations on Conway's Game of Life and Other Cellular Automata. Research Paper Presented to the Students and Teachers as Research Scientists Program at the University of Missouri-St. Louis Sponsored by LMI Aerospace, Inc./D3 Technologies and The Solae Company.

[5] Fujita, T., Nakano, K., & Ito, Y. (2016). Fast simulation of Conway's Game of Life using bitwise parallel bulk computation on a GPU. International Journal of Foundations of Computer Science, 27(08), 981-1003.

[6] Weeden, A. (2013). Parallelization: Conway's game of life.

[7] Ma, L., Chen, X., & Meng, Z. (2012). A performance Analysis of the Game of Life based on parallel algorithm. arXiv preprint arXiv:1209.4408.

[8] Jahan, M. V., & Khosrojerdi, F. (2016). Text Encryption Based on Glider in the Game of Life. International Journal of Information Science, 6, 20-27.

[9] Knafla, B., & Leopold, C. (2008). Parallelizing a real-time steering simulation for computer games with OpenMP. Parallel Computing: Architectures, Algorithms, and Applications, 15, 219.

[10] Lee, K. M., Song, T. H., Yoon, S. H., Kwon, K. H., & Jeon, J. W. (2011, October). OpenMP parallel programming using dual-core embedded system. In 2011 11th International Conference on Control, Automation and Systems (pp. 762-766). IEEE.

[11] Akhmetova, D., Iakymchuk, R., Ekeberg, O., & Laure, E. (2017, May). Performance study of multithreaded MPI and OpenMP tasking in a large scientific code. In 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) (pp. 756-765). IEEE.

[12] Chauhan, K., Bhensdadia, C. K., & Potdar, M. B. (2017). Parallel Computing Models and Analysis of OpenMP Optimization on Intel i7 and Xeon Processors. International Journal of Computer Science and Software Engineering, 6(12), 315-322.

[13] Wang, A. I., & Wu, B. (2015). The Use of Game Development in Computer Science and Software Engineering Education. Computer Games and Software Engineering, CRC Press, Taylor & Francis, Boca Raton, FL (to appear).

[14] Kultima, A. (2015, September). Game design research. In Proceedings of the 19th International Academic Mindtrek Conference (pp. 18-25). ACM.

[15] Wu, G., & Tao, J. (2016, May). Design and improvement of the parallel algorithm in the computer games system. In 2016 Chinese Control and Decision Conference (CCDC) (pp. 6884-6887). IEEE.

[16] Zamith, M., Valente, L., Feijó, B., Joselli, M., & Clua, E. (2015, November). Exploring parallel game architectures with tardiness policy. In 2015 14th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames) (pp. 33-42). IEEE.

[17] Asaduzzaman, A., Lee, H. Y., & Gummadi, D. (2014). Impact of Thread Synchronization and Data Parallelism on Multicore Game Programming. In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA) (p. 1). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

[18] Andblom, R., & Sjöberg, C. (2018). A Comparison of Parallel Design Patterns for Game Development.

[19] https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

[20] https://www.geeksforgeeks.org/openmp-hello-world-program/

[21] https://www.geeksforgeeks.org/mpi-distributed-computing-made-easy/

[22] https://www.gamedesigning.org/career/programming-languages/

[23] https://stackoverflow.com/questions/33258762/how-to-mix-serial-and-parallel-code-using-mpi

[24] https://link.springer.com/book/10.1007%2F978-1-84996-217-9

[25] https://books.google.com/books?hl=en&lr=&id=z-WoCAAAQBAJ&oi=fnd&pg=PR7&dq=cellular+automata+game+of+life&ots=XAN0sz19r5&sig=glcGLVlZGtX1KBAIUunNyMfoXpE

[26] http://web.mit.edu/sp.268/www/2010/lifeSlides.pdf

[27] http://www.scholarpedia.org/article/Game_of_Life

[28] https://www.researchgate.net/post/Which_parallelising_technique_OpenMP_MPI_CUDA_would_you_prefer_more