Parallel Conway’s Game Of Life

Bhramar Choudhary
Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai, India

150050013
bhramar@iitb.ac.in

Deepak Meena
Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai, India

150050039
150050039@iitb.ac.in

Deepesh Meena
l *Computer Science and Engineering*
Indian Institute of Technology, Bombay
Mumbai, India

150050041
150050041@iitb.ac.in

Anshu Ahirwar
Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai, India

150050077
1500050077@iitb.ac.in

***Abstract*—Using OpenMP to parallelise the zero player game Conway’s Game of Life, which requires no further input than its initial state**

Keywords—OpenMP, Game of Life

I. INTRODUCTION

Game of Life:

Motivated by questions in mathematical logic and in part by work on simulation games by Ulam, among others, John Conway began doing experiments in 1968 with a variety of different 2D cellular automaton rules.

Conway’s initial goal was to define an interesting and unpredictable cell automaton.The game made its first public appearance in the October 1970 issue of *Scientific American*, in Martin Gardner's "Mathematical Games" column. Theoretically, *Conway's Life* has the power of a universal Turing machine: anything that can be computed algorithmically can be computed within *Life*.Since its publication, *Conway's Game of Life* has attracted much interest, because of the surprising ways in which the patterns can evolve. *Life* provides an example of emergence and self-organization. Scholars in various fields, such as computer science, physics, biology, biochemistry, economics, mathematics, philosophy, and generative sciences have made use of the way that complex patterns can emerge from the implementation of the game's simple rules.

OpenMP:

OpenMP (**Open Multi-Processing**) is an application programming interface (API) that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran,[3] on most platforms, instruction set architectures and operating systems, including Solaris, AIX, HP-UX, Linux, macOS, and Windows. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior

II.Rules And Implementation

A.Rules In Game Of Life:

The universe of the *Game of Life* is an infinite, two-dimensional orthogonal grid of square *cells*, each of which is in one of two possible states, *alive* or *dead*. *Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:*

1. *Any live cell with fewer than two live neighbours dies, as if by underpopulation.*
2. *Any live cell with two or three live neighbours lives on to the next generation.*
3. *Any live cell with more than three live neighbours dies, as if by overpopulation.*

4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

The initial pattern constitutes the seed of the system. The first generation is created by applying the above rules simultaneously to every cell in the seed; births and deaths occur simultaneously, and the discrete moment at which this happens is sometimes called a tick. Each generation is a pure function of the preceding one. The rules continue to be applied repeatedly to create further generations.

B. Serial Version:

The program accepts a file and number of iterations desired as inputs. The initial state is then read from a file. Alternatively, a different file takes no inputs: the desired grid size and iterations must be altered manually in this file, and it generates a random grid of said size. The rules of the game are then applied to the initial configuration and the subsequent configurations. The main function calls upon several functions that populate the grid space, find the number of neighbours alive and so on. The updates to the grid space are done via if conditional statements and two 2D integer arrays.

C. Parallel Version:

The program parallelises the various for loops in the serial version of the game. Since the program relies heavily on operations on a 2D integer grid, this lends a lot of opportunities to use openMP, where omp for is used to split up loop iterations among the threads

III. Some Notes

From most random initial patterns of living cells on the grid, observers will find the population constantly changing as the generations grow. Small isolated sub patterns with no initial symmetry tend to become symmetrical. Most initial patterns eventually burn out, producing either stable figures or patterns that oscillate forever between two or more states. many also produce one or more gliders or spaceships that travel indefinitely away from the initial location. Many patterns in the Game of Life eventually become a combination of still lifes, oscillators, and spaceships; other patterns may be called chaotic. A pattern may stay chaotic for a very long time until it eventually settles to such a combination.

IV. Graphs and Observations

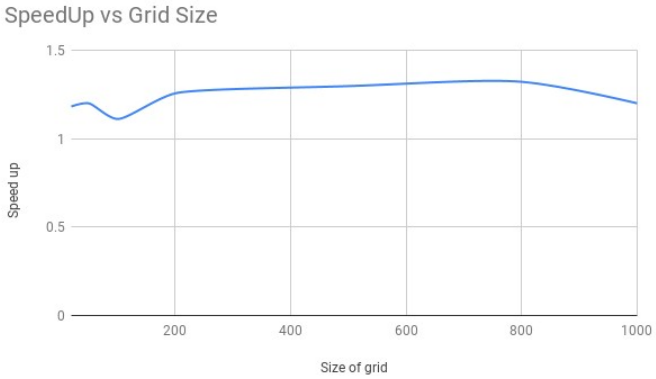

Illustration 1: Speedup with respect to size of grid

Table For above data:

Size of Grid	Speedup
2	1.1818182
50	1.2
100	1.11111
200	1.25622776
500	1.2971
800	1.3215
1000	1.19996

- The speedup is obtained by dividing the time of serial version with parallel version

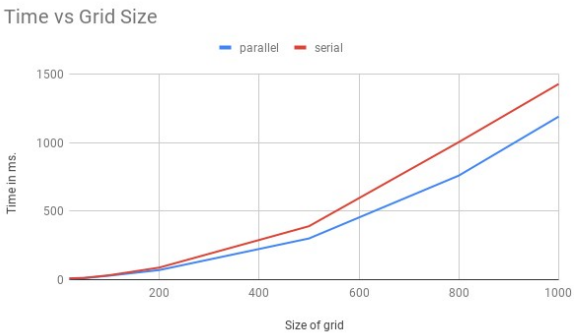

Illustration 2: Comparison Of serial and parallel with respect to Grid Size

- As we can see as we increase the size of grid , the difference between time taken by the serial version and parallel version increases.
- This is because we have parallelise over the size of grid so as grid size , the better our code works

Illustration 3: Time taken by serial and parallel w.r.t iterations

-Code is parallelised over size of grid.In every iteration , code iterate through grid size . So parallel verion of

will be faster comapred to serial as each iteration will

be done quicker .

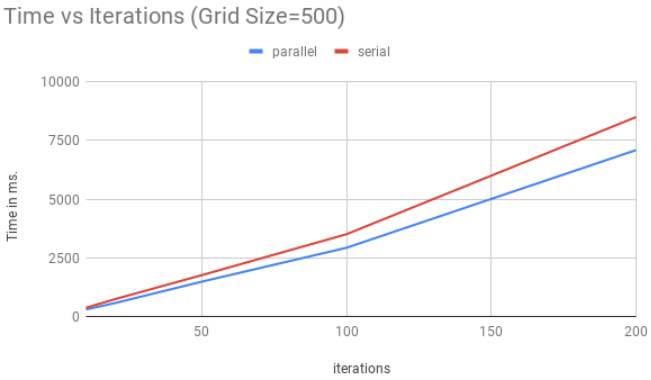

Illustration 3: Serial and parallel performance against no. of iterations

-Theoretically as no. of threads increases time should decrease but opposite to that when threads increases from 4 to 6 time almost remain . Maybe this observation will ne machine specific

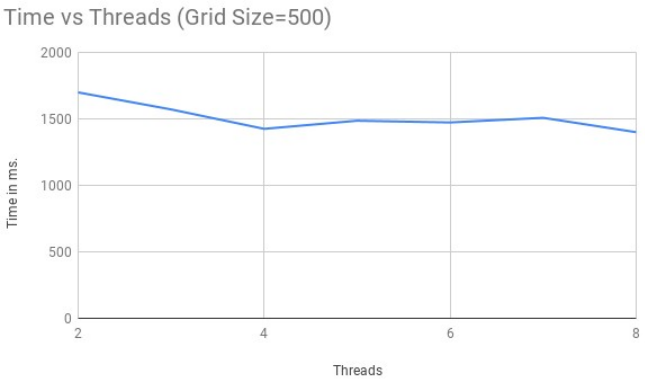

Illustration 4: Performance of parallel version against no. of threads