Conway's Game of Life using MPI

Ellyess Benmoufok
efb119@ic.ac.uk

February 23, 2020

Abstract

This report documents C++ code written to simulate Conway's Game of Life, using MPI to parallelise the code for a large decrease in run time. Using MPI the run-time was able to decrease a lot for a 10000 by 10000 grid for game of life. For periodic boundary conditions there was up to a 98% decrease in run time by using 216 cores as opposed to 1 core.

1 Introduction

Conway's Game of Life is a cellular automation, that you provide the initial state to and watch it evolve, it was devised by British mathematician John Hornton Conway. Life can be coded in C++ effectively but can be slow for large grids in serial code. MPI allows communication between multiple cores where each core can work on a specific grid of Life. The way the parallelised code has been written is in a way that it devises how the total grid of game of life can be split over any amount of cores provided to run the program.

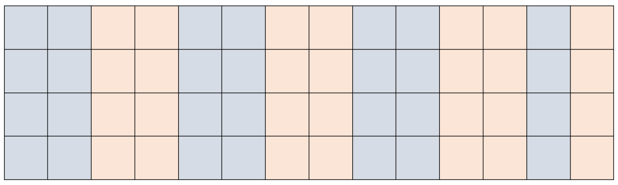

Figure 1: How 9 processors are defined for a 16 by 4 Game of Life grid

In my Life.cpp file the function **find_dimensions** devises the best way to organise the processors in a way that each processor sub-grid is as close to square as possible as displayed in Figure 1. This is method is used to provide efficient communications. MPI will track the dimensions of each processor and using 16 different data types 8 for sending and 8 for receiving as displayed in Figure 2, the communications ensure the corresponding data types are sending and receiving at each step.

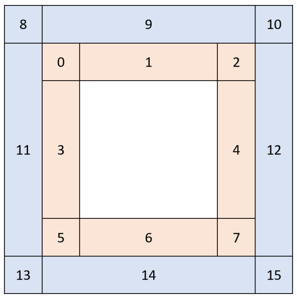

Figure 2: Processors communication DataTypes; red is send, blue recieve.

2 Performance and Results

In order to test the performance of the parellelised Life code, I ran various grid sizes on Imperials HPC with varying cores. I timed performance on the following cores 1, 12, 24 then intervals of 24 until 216 cores.

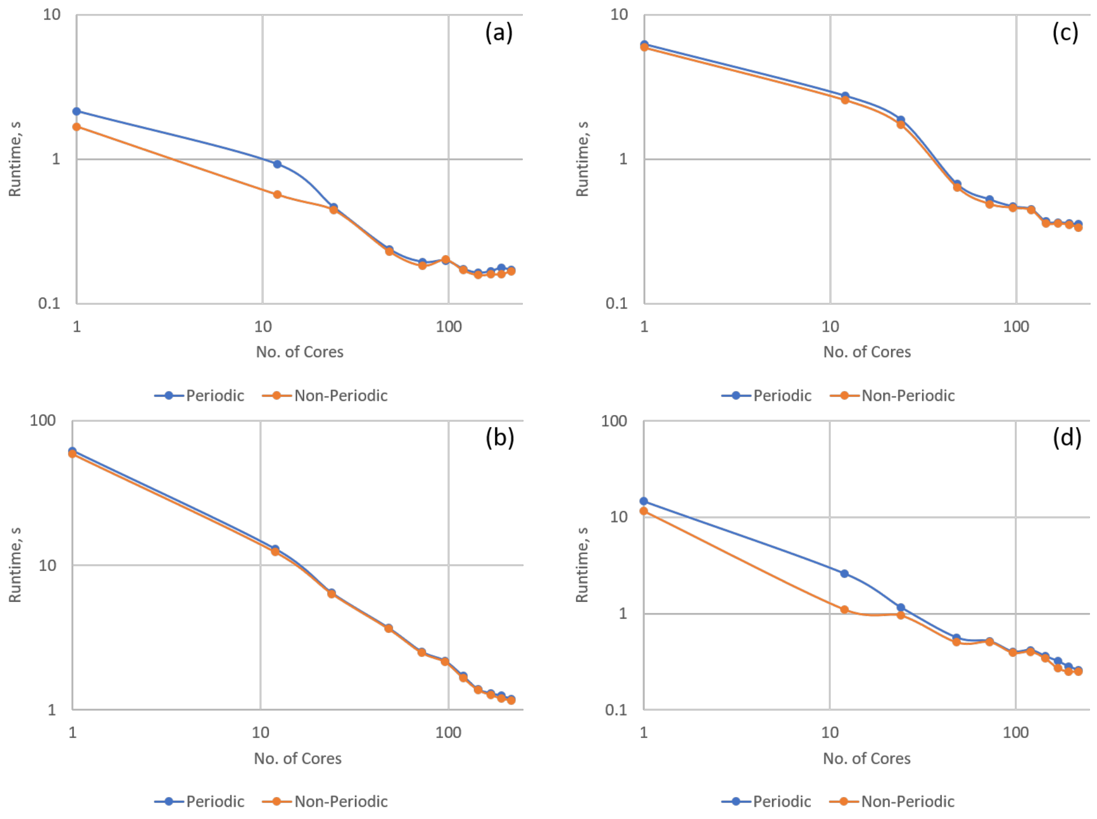

Figure 3: The run time vs. cores for (a) 1000x1000, (b) 10000x10000, (c) 1000x10000 and (d) 10000x1000

For a fixed period of 100 iterations I ran Life on these grid dimensions, 1000 by 1000, 10000 by 10000, 10000 by 1000, 1000 by 10000. The general trend pre-testing I expected to see a large increase in speed from serial code to parallel running with many cores, with the increase levelling off at a certain point, thus I decided to use a log-log plot format for the results to see a clearer trend in speeds for the larger number of cores.

Periodic vs Non-Periodic

Naturally we would expect a longer run-time for periodic boundary conditions as it requires more communications between the cores, however when running test for both periodic and non-periodic there was no noticeable difference in time, this can be identified in Figure 3. Figure 3(a) and (d) show faster run-times at lower number of cores for non-periodic.

Game of Life Grid Size

My initial thought is that the Life grid would increase the run-time as it increases and this is clear from Figure 3(a) compared to Figure 3(b) which show a base 10 difference in timings for the larger grid. A noticeable feature the log-log plots show is that the a some of the grids doesn't have the best linear line on the plot I recognised this may be due to the grids being used being small in size so the speed increase from MPI can vary between runs. On the other hand, Figure 3(b) which is the largest grid shows a close to linear trend which

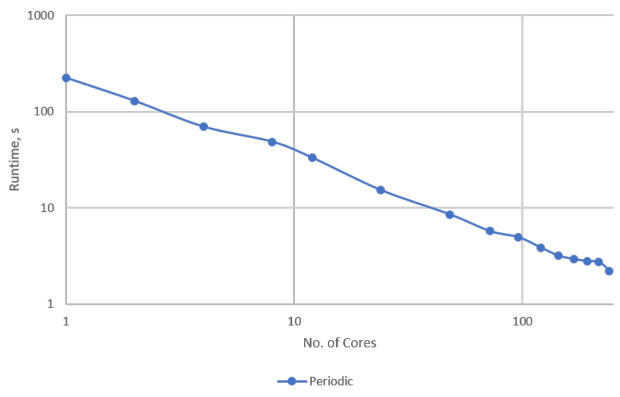

Figure 4: The run time vs. cores for a 20000 by 20000 grid with 50 generations

led me to the thought that the larger the grid the more linear the trend. To further support this thought I tested a larger grid for this purpose of 20000 by 20000 for 50 generations which is displayed in Figure 4, the trend is close to being a straight line.

Efficiency and Speed Up

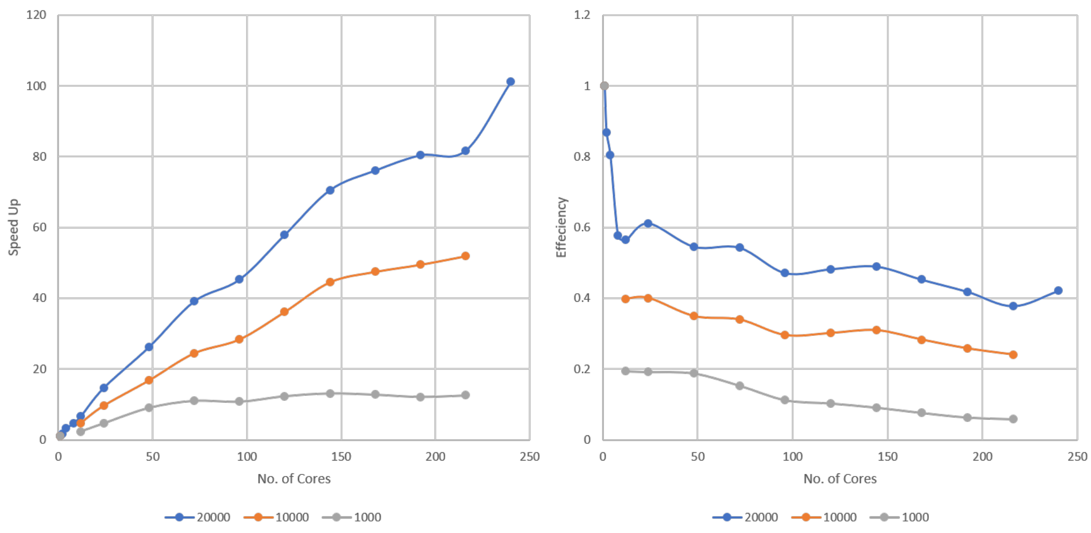

Figure 5: (a) Speed Up and (b) Efficiency for 20000 by 20000, 10000 by 10000 and 1000 by 1000

The speed up of the program shows a larger effect on larger grids for Life as displayed in Figure 5(a) being closer to linear at the 20000 by 20000 grid. This leads to the efficiency which ideally should have a linear trend from an efficiency of 1 as the cores increase, however Figure 5(b) shows that all the grid sizes have a drop then continue in a linear manner. This is an issue with how I've timed my program as it is also timing the initialisation of all the randomly generated game of life cells and all other functions before the start of the parallelised game of life.

3 Discussion and Conclusions

In conclusion the results were as expected and followed a clear trend you would expect when using more cores to do the work. The trends discovered make it clear that MPI is best used for programs requiring larger data processing, such as the grids in game of life that were above 10000 by 10000, as they have increasingly better speed up. I have learnt how to communicate between processors using multiples cores and can see how this would be very useful for programs working with large amounts of data and large arrays. What could have been improved the performance testing is using repeat timings of each data point taken, as well as improved placement of the timer. Also I would want to compare how using booleans may be better than integers for Conway's Game of Life; and trying different methods of communications such as using data types that specifically send columns and rows instead of predefined shapes.