Nicolas Trinephi
01212102

23/02/2020
ACSE-6

ACSE-6 Patterns for Parallel Programming

Game of Life

Conway's Game of Life is a classic automaton based on very simple rules which dictate whether a cell lives or dies in the next round of the "game". Different rules and geometries can also be implemented to add extra complexity to the game. In this implementation, the classic rules were used: a living cell dies with 1 or less live neighbors and 4 or more live neighbors and continues living with 2 or 3 live neighbors; a dead cell becomes alive if it has 3 live neighbors. The interest of this study was to implement the game of life in parallel to be ran on multiple cores simultaneously and to observe the performance with different grid sizes and number of cores. This was completed using Microsoft's MPI library in C++ and Imperial College London's High-Performance Computing (HPC) system.

Implementation:

The Game of Life code was written using the Eclipse IDE and features the grid implemented as a Life class with methods performing various tasks necessary to execute the game in parallel. A class was chosen here because every game has specific parameters and a class enables them to be grouped into a single object. This increases clarity of the code and reduces the risk of confusion and errors. Communication between processors are completed peer to peer which allows each processor to have a similar workload, increasing efficiency of the code. Furthermore, each processor creates its own grid and random disposition of live and dead cells. A master processor in control of the program is therefore unnecessary for completing game iterations.

When considering grid dimensions and individual core domain dimensions, the most optimum setup is to have each core take responsibility for a domain that is as close to square as possible rather than having a square arrangement of processors. This is beneficial because it minimizes the amount of data required to be communicated thus lowering communication overhead. This thought process is accomplished in the code and allows cores to have square-like domains even for excessively narrow grids whenever possible.

Nicolas Trinephi 23/02/2020
01212102 ACSE-6

<u>Performance Analysis</u>:

All tests were done on the HPC while using various number of cores and corresponding nodes.

Tests were completed on various square grids (figure 1) and a couple rectangular narrow grids (figure 2) to explore how these changes would affect performance. In all cases, only processor 0 time was measured a few times and averaged, Origin Lab Pro could have been used to display error bars however I was unable to download it through Imperial after many attempts. All times are displayed in figure 3. Taking measurements using any other processor to represent the performance of the code would yield similar results. The number of cores is expected to affect performance in that generally a larger number of cores decreases performance time because workload is distributed. However, some special cases may emerge.

For the square grids, the domains of each processor can easily be divided into squares, so it is expected for time to decrease with an increase in processor number. This is exactly what is observed in the test times. Times steadily decrease until the decrease is negligible after around 70 cores. But we also observe that execution time rises again after exceeding a certain number of cores. While distributing tasks is generally a good idea, the time that processes take to send each other data must not be ignored. This overhead becomes more evident for smaller grids when processor oversaturation occurs for lower numbers of cores. For the 5 000 and 10 000 side size square grids, even pushing past 280 cores, there is not yet any noticeable overhead which decreases performance. For the 1000 and 500 side square grids, the overhead is more noticeable starting at 150 and 100 cores respectively.

When looking at the speed up ratios and parallel efficiency graphs in figure 1 and figure 2, we notice trends which are expected. As core number increases, the efficiency diminishes until it plateaus, this is expected because while work load between processors is spread out, they must send data to one another, and the more processors there are, the more data is sent and received. However, the larger grids do plateau at a higher time but at around the same number of cores. This means that the plateauing is dependent on cores, the workload simply dictates how long the execution takes. Similarly, the speed up ratio does behave as expected and increases with core number. And similarly, again, the effect is more visible on larger grid sizes. The smaller grids plateau at smaller core number as mentioned above with the overhead and saturation above.

To further test the effects of the communications on performance, non-square grid sizes were also used on the HPC. The figures are presented in figure 2. A 10 000 by 100 grid has the same number of cells as a 1000 by 1000 grid, but it is observed that performance is quite different. The elongated grid has less efficiency at higher core numbers as well as dips in speedup ratio. This can be explained by the narrow arrangement of the grid which forces cores to take elongated non-square domains. And at very high core numbers where core domains are small enough to be square, the communication overhead takes over and the efficiency remains low. Exchanging the row value for columns as C++ is a row major order language does not change performance to a very noticeable degree.

For the 5k x 10, low number of cores or high number of cores perform similarly with regards to speed up ratio and the efficiency plateau is met at only 12 cores. This due to the arrangement of the grid: it is very narrow. This configuration makes it difficult to have square domains unless the domain is extremely large compared to the narrow side or unless the domain is on the same order of magnitude as the narrow side (i.e. 2500 x 5 domains or 2 x 2 domains for example).

Overall, the periodic and non-periodic times differ very little although it is expected for the periodic boundary games to take slightly longer than non-periodic games simply because the amount of communications between processors is greater.

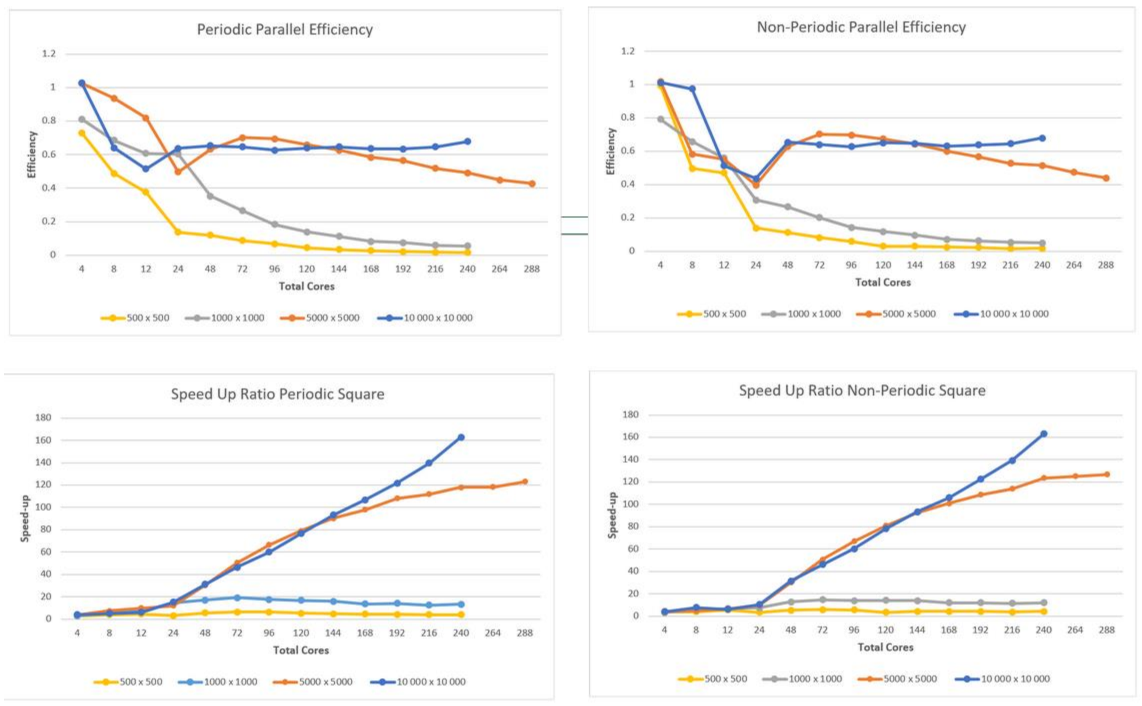


Figure 1: Parallel Efficiency and Speed Up Ratio for Square Arrangements

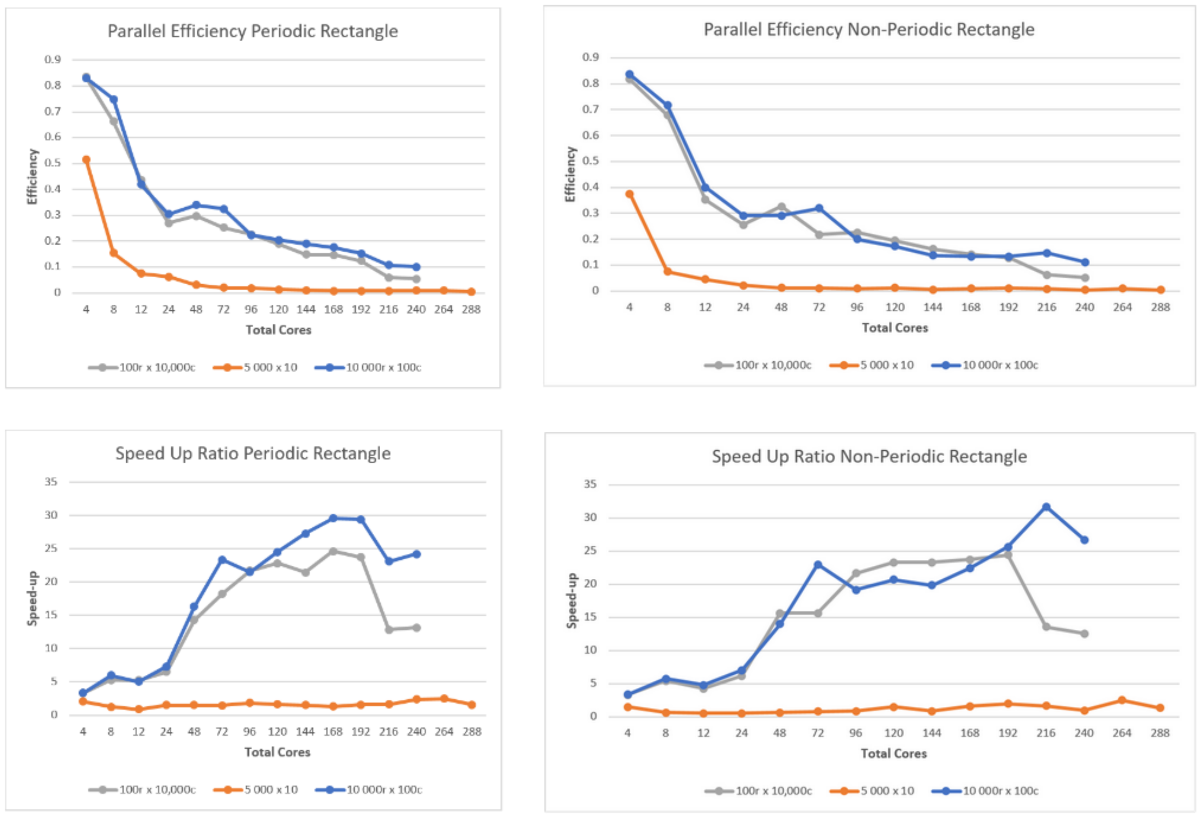
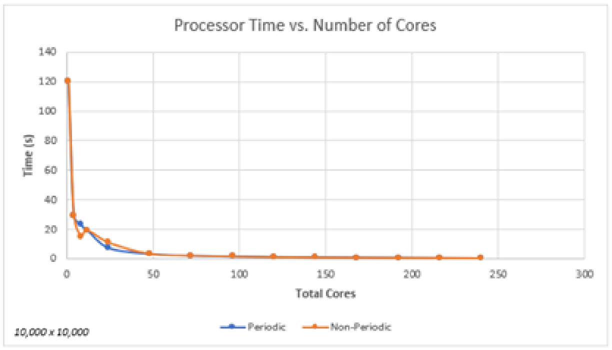
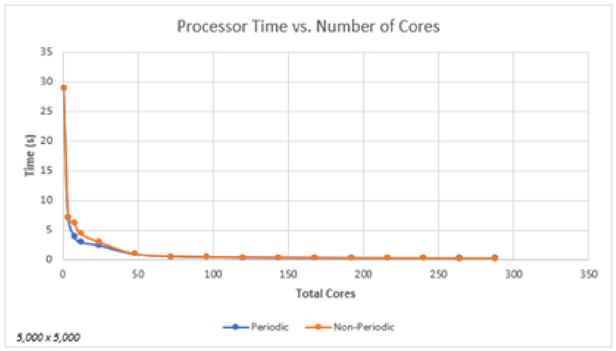


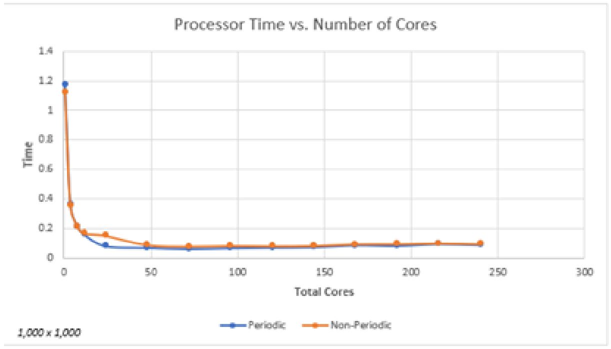
Figure 2: Parallel Efficiency and Speed Up for rectangular arrangements

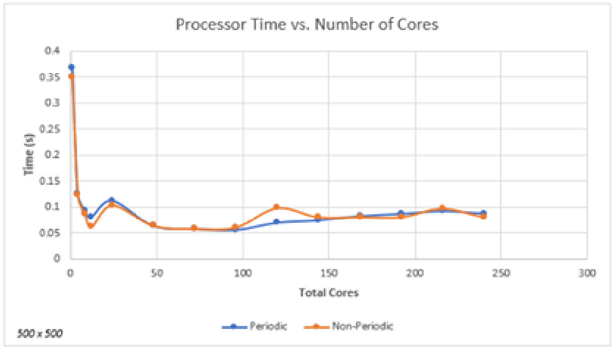
Nicolas Trinephi
01212102

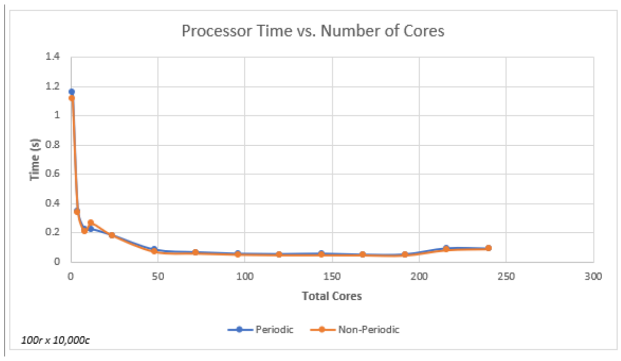
23/02/2020
ACSE-6

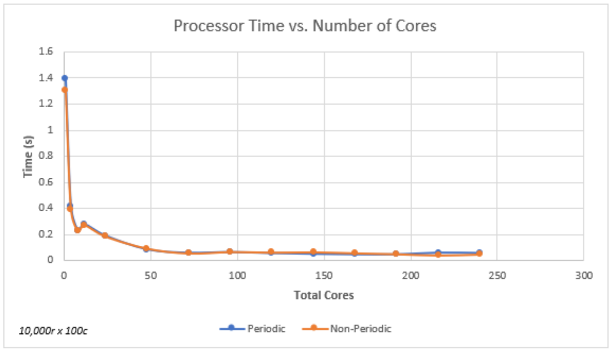












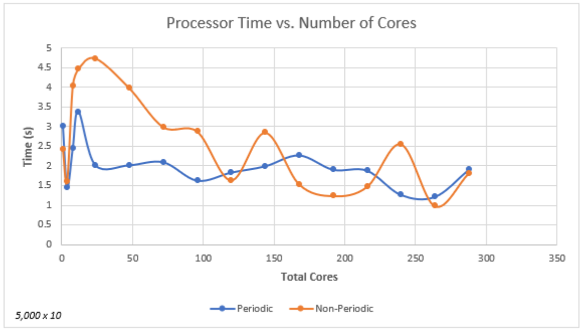


Figure 3: Execution Times for different configuration of grids with different number of cores