Prepared for submission to University of Cambridge

C2: Advanced Research Computing - Coursework Assignment

Adnan Siddiquei

University of Cambridge

E-mail: as3438@cam.ac.uk

Contents

1 Introduction

Conway's Game of Life...

2 Selection of Solution Algorithm and Prototyping

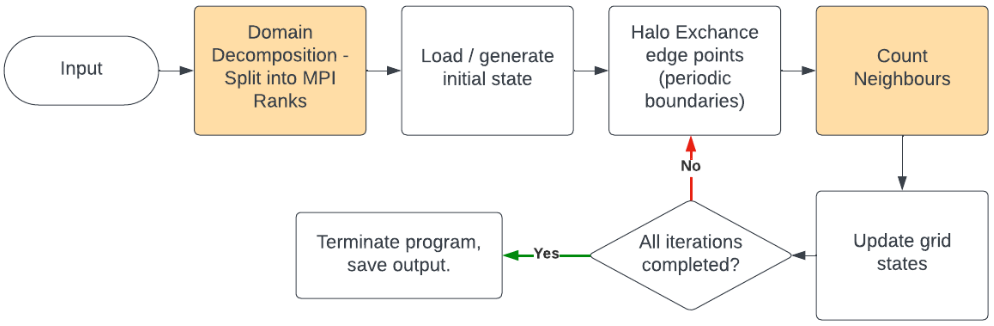
Input
Domain Decomposition - Split into MPI Ranks
Load / generate initial state
Halo Exchance edge points (periodic boundaries)
Count Neighbours
Update grid states
All iterations completed?
No
Yes
Terminate program, save output.

Figure 1: A high-level flow chart depicting the algorithm. The key points for efficiency considerations are highlighted in orange, the considerations are discussed in the main text.

Fig.(1) shows the high-level algorithm being used to simulate Conway's Game of Life. The key points of consideration for efficiency are highlighted in orange. It is also worth noting that the rest of this section assumes that the grid is stored as a 1D array in row-major order.

Domain decomposition can be done by either strips or blocks. Strips has the advantage that it is simpler to implement but will have more memory needing to be passed between boundaries, with the opposite being true for blocks. In the case of the neighbour counting algorithm, there are a couple of possible implementations.

- **Simple convolution.** Loop through each cell, and count the number of neighbours. This is equivalent to a 3x3 convolution with the below kernel.

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.1}$$

  This requires a total of $8N^2$ addition operations (the $9N^2$ multiplication operations that would usually occur can be ignored because the kernel only contain 1s and 0s so we can skip the multiplication operation in this special case).

- **Separable kernel.** If the following kernel is used instead,

$$\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \tag{2.2}$$

then the kernel becomes separable. This means that the kernel can be split into two 1D identical kernels $\begin{bmatrix}1 & 1 & 1\end{bmatrix}$ and applied in two passes which yields $2N^2$ addition operations per pass, yielding a total of $4N^2$ operations [1]. After another N^2 addition operations, to correct for replacing the central 0 with a 1, the total number of operations becomes $5N^2$. Therefore, the separable kernel method yields a theoretical speedup of $8/5 \approx 1.6$.

The simple convolution method introduces the issue of strided memory access when counting the 6 neighbours that are not horizontally adjacent to the current cell, as illustrated in Fig.(2).

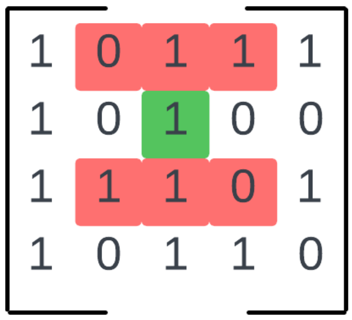

Figure 2: Illustration showing that the 6 vertical neighbours (highlighted in red) of the green cell are not contiguous in memory when representing a 2D array as a 1D array of points in row-major order.

Therefore, when implementing the simple convolution method, column-wise decomposition would be preferred as it would reduce the number of cache misses. This is because an $N \times N$ grid split into P columns would yield a `N/P * sizeof(int)` byte distance between a cell and it's vertical neighbour whereas the same grid split into P rows would yield a `N * sizeof(int)` byte distance, and P blocks would yield a distance of yield `N/sqrt(P) * sizeof(int)` (assuming P is a square number and N is perfectly divisible by P).

However, the separable kernel method does not have this issue as the data is contiguous on the horizontal pass, and the data can be made contiguous for the vertical pass by transposing the grid. Therefore, it does not matter which decomposition method is used for the separable kernel method with respect to cache misses. Furthermore, the type of domain decomposition used will also affect the speed of the transpose operation. If the transpose operation is done by writing contiguous and reading strided (which is generally faster), then column decomposition would be preferred as it would reduce the number of cache misses when reading the grid.

However, which decomposition method is faster is still dependent on how the memory exchange rates between MPI ranks affects the overall performance of the algorithm, meaning both columns and blocks will need to be explored for efficiency. Likewise, a variety of OpenMP techniques will be used to multithread the convolution and transpose operation within each MPI rank, and various single thread optimisations will be explored to make use of SIMD vectorisation on the CPU level. These are all explored and the assumptions above are all tested in the next section to identify the optimal algorithm.

References

[1] Steve Eddins, *Steve on Image Processing with MATLAB*. Available at: https://blogs.mathworks.com/steve/2006/10/04/separable-convolution/ [Accessed: 8-Mar-2024].