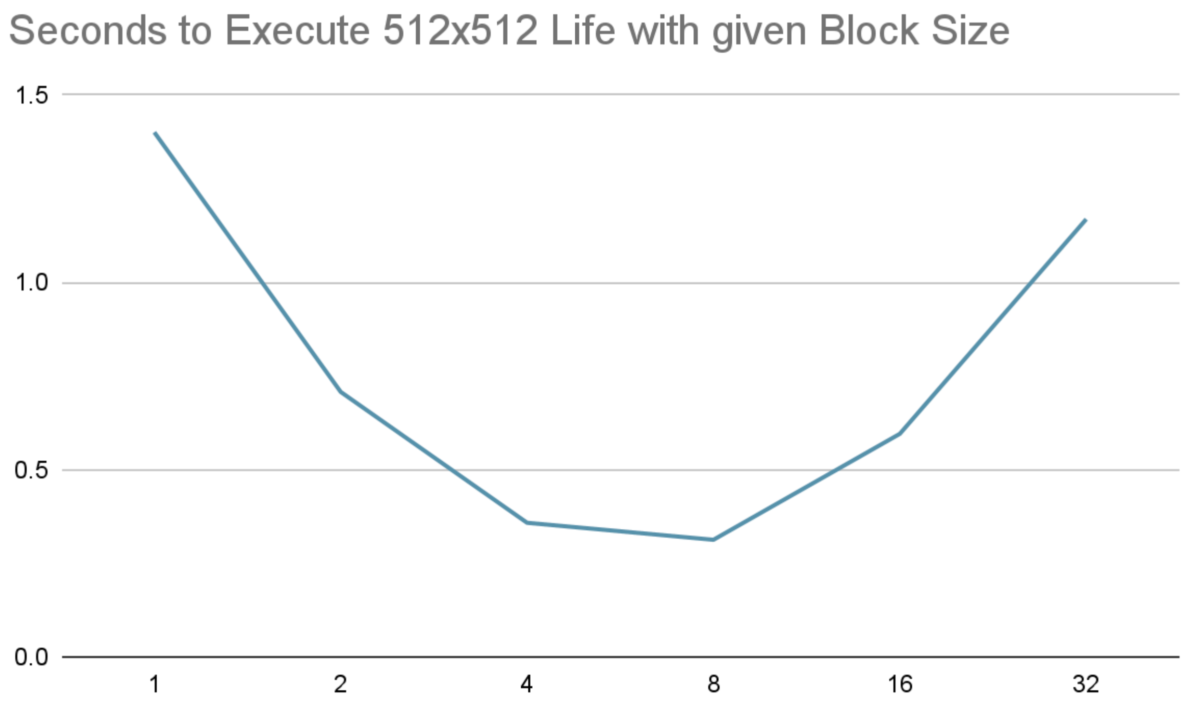
Writing the kernel was not bad at all. The two functions, padded_matrix_copy and and compute_on_gpu both were quite similar to their serial counterparts, just requiring slightly altered formulas using the initial value for i for a given core and then incrementing by stride. Both have essentially identical workflows to the serial code, evident by most of the code being directly copy and pasted, with some calculations and loops changed.

The initial data distribution was done using initial i, which is calculated using blockDim.x * blockIdx.x + threadIdx.x. This formula was used for both kernel functions. This allows every kernel to understand where exactly it is and where it should begin work. After that, the index is incremented with striding, which is calculated using blockDim.x * gridDim.x. This ensures that no cores double dip and do the same work.

Below are the performance results of running the program on different block sizes.



The results show that the optimal block size for 512x512 game of life is 8x8. While I do not totally understand why this is the best, it is likely just the best combination of communication and computation, as most optimal solutions in this class are. Depending on the size of the life grid, the optimal block size likely changes as well, with smaller games working best with small block sizes, and larger games working best with large block sizes. Overall, these results seem pretty in line with most other findings in parallel computing, being more or less what I expected.