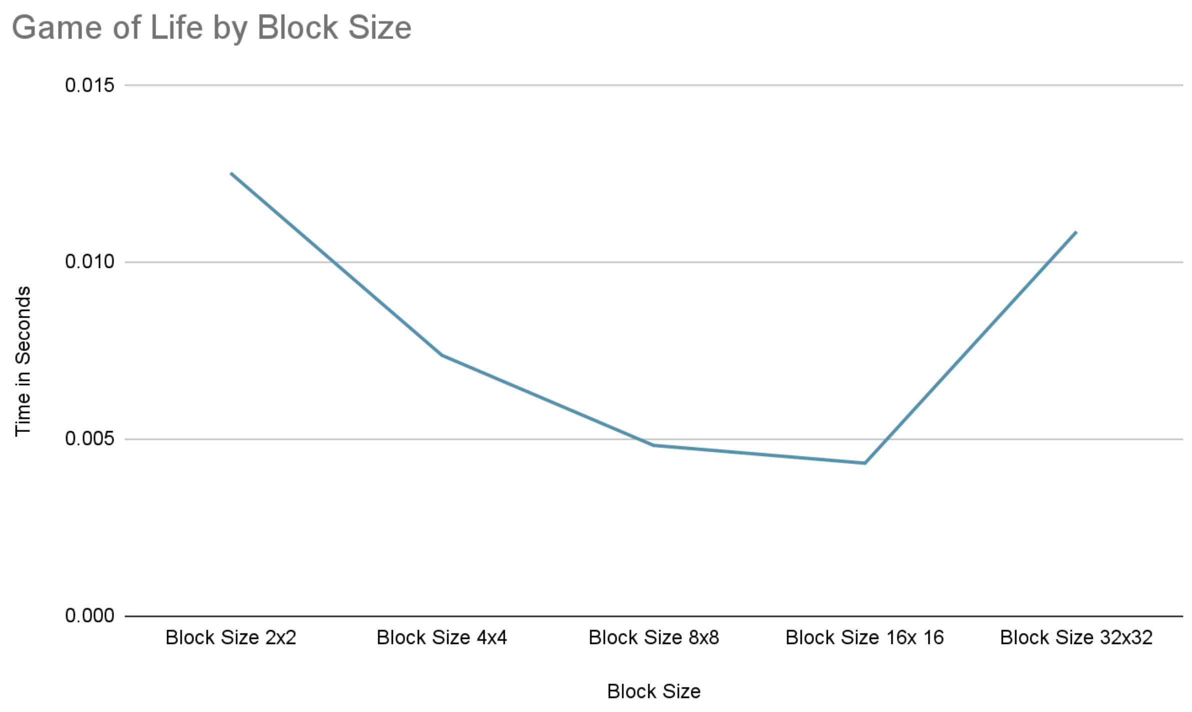

The kernel functions I wrote were the padded_matrix_copy function and the compute_on_gpu function. In these functions I used a function to match the previous_life and life matrices. Since the regular life matrix is made into 1 long function I had to use this function to get the regular index of life:

$$X\_index * width + Y\_index$$

This is because X_index is the row that the array element is in and you multiply it by the width to get to the first number in the row. Then all you have to do is add the Y_index to get to column of the element we want. Then I used this function to match it up with previous life matrix:

$$(i + padding) * (width + padding) + j + padding + padding$$

This is because the previous life matrix has a padding of 1 that is later used on in the compute_on_gpu function. I basically use the same original equation above but add the proper padding involved to line them up.

As for writing to the correct threads, I had to use a couple equations to get the proper thread column and row. This being:

threadId.x + blockId.x*blockDim.x
threadId.y + blockId.y*blockDim.y

I used the threadId.x and threadId.y to understand the correct columns and rows in a block, then used blockId.x*blockDim.x and blockId.y * blockDim.y to get the right block that the thread needed. What this does is gets the thread so that it can only process one block at a time, which would be the process that correctly matches up with the grid of threads. Then I had to add striding which I did by multiplying the block dimension by the grid dimension so that each thread would go past the entirety of the grid to get to its next element that it has to process.

After using all the equations above all I had to do was use the original two equations to find the neighbors of the element we are trying to process and check if the element we had was dying from overpopulation or loneliness, being reborn, or surviving.

To distribute the data all I did was use the cudaMemcpy with a copy of the array, an empty array and the CudaMemcpyHostToDevice variable. This copied the array to the empty array only in the thread space. The arrays get copied back after computation by doing the same thing except switching CudaMemcpyHostToDevice to CudaMemcpyDeviceToHost

The data I collected on the plot graph are what I expected except for the end. I didn't expect the 32x32 block to be higher than the previous two before it. That being said, it is inline with the previous projects we have done. With the bigger size of blocks came more communication within the blocks, just like the threads in our previous projects.