Project 4 Report

Christopher Chen, Kyungbong Ko

Gaussian Blur Serial

Much of the initialization code was adapted from file heat.c of project 2 and from the mandelbrot serial implementation. This included the parsing of CLI arguments and parts of saving the result into the appropriate output file. In this project specifically, we followed the PGM format and wrote to the file accordingly. This project also required us to read a given PGM file as well, which we implemented with the fgets() function.

The next step was to make the kernel matrix, which was done in 2 functions, create_kernel_matrix() which uses guassian_func(). Guassian_func() calculated each value of the matrix given the equation, and upon testing, we realized that the floating point error was within the bounds for error checking so we decided not to normalize the values. For create_kernel_matrix(), we initially defined the center of the kernel as having the coordinates (0, 0), which had good intentions but resulted in the values being centered around the start of the allocated memory, leading to undefined behavior.

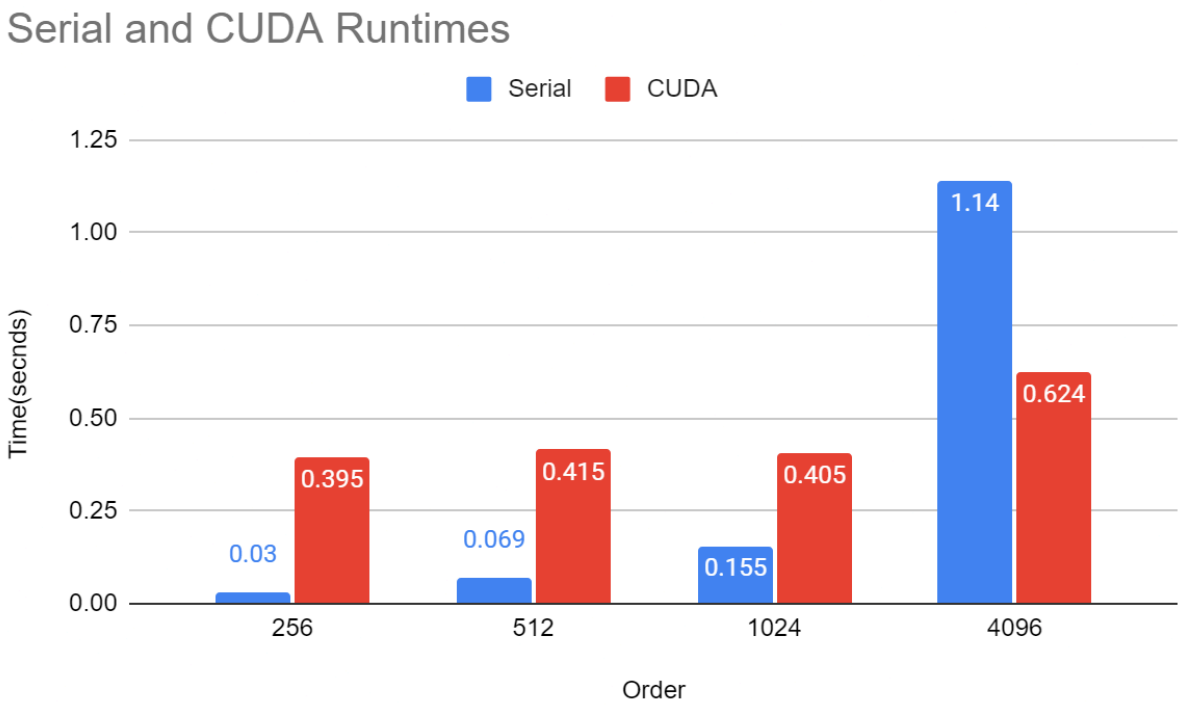
After recentering the kernel matrix, guassian_blur() handled the blur calculations. Something noteworthy here is we had a helper function that handled the cases where the kernel locations were outside the bounds of the image.

Gaussian Blur CUDA

The main difference between the serial and the CUDA versions is the setup required for the SM cores to execute the kernel, which each received a section of pixels to calculate the blur with the kernel matrix.

A 2-D grid was used, with each block having an order of 32 by 32 threads for a total of 1024 threads, the maximum number of threads available for a block. The number of blocks were

calculated using this value, and after allocating space on the GPU for the input and output images along with the kernel, the kernels were launched.



As for results, we saw that for smaller orders the serial version overwhelmingly outperformed the CUDA versions(~10 times speedup). However, we can also see that once the order increases to 4096, the time taken for the serial version increases tenfold, pointing to an exponential or at least quadratic increase in the runtime, whereas the runtime for the CUDA version increases by a factor of 1.5. This trend is expected to continue as the order increases and the effect of parallelization becomes more and more prominent. We suspect that the initial slowdown for the CUDA comes from the overhead in allocating memory and initializing the blocks/grids.