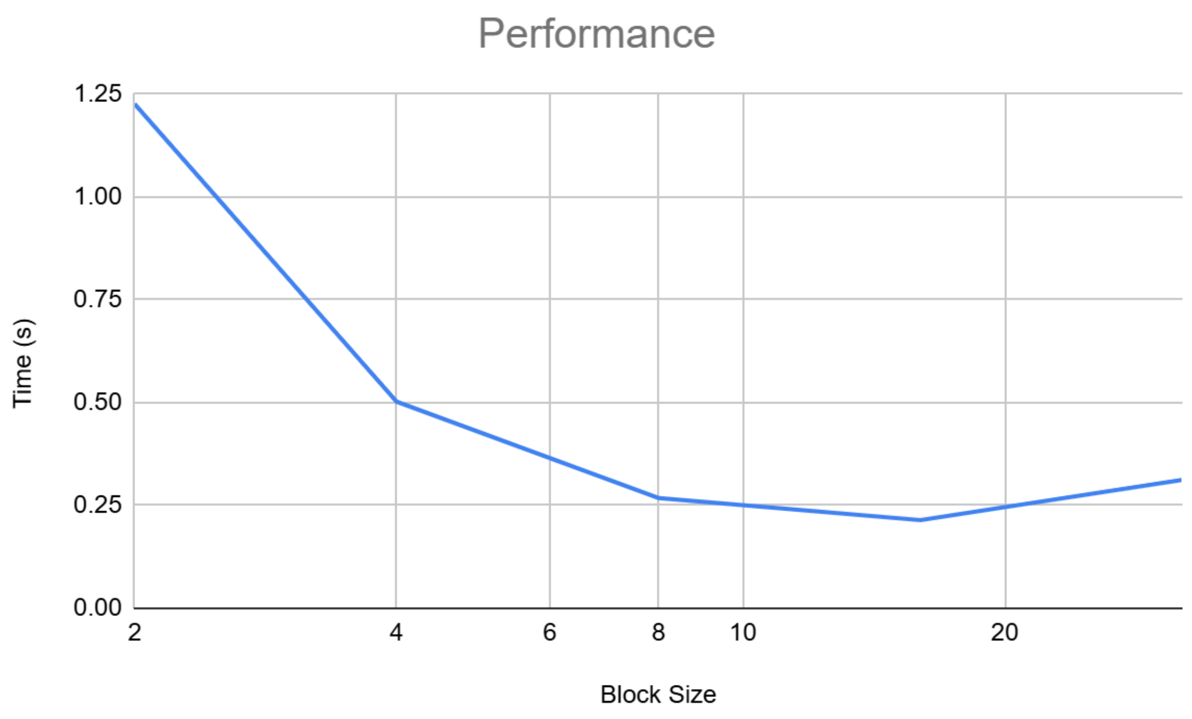
Kernel Implementation: The convolveGPU kernel was implemented by mapping each thread to process multiple pixels in the image using striding. The global thread indices were calculated based on the block and thread indices to determine each thread's starting pixel. To ensure full image coverage, especially when there are more pixels than threads, I increment the row and column indices by the total number of threads in the grid (strideX and strideY). Each thread checks if the current pixel is within the valid convolution range. This avoids border pixels. For each valid pixel, I performed the convolution by iterating over the kernel dimensions, computing the dot product separately for each color channel. The convolution results were then written back to the corresponding location in the output image.

Initial Data Distribution: The initial data distribution transfers video frames from the host to GPU. There is a batching strategy where a batch of frames are loaded into memory, and memory is allocated on the GPU for both input and output frames. Each frame is copied to the GPU using cudaMemcpy. This ensures that data transfer overlaps with computation whenever possible. CUDA streams enable concurrent execution of memory transfers and kernel computations to maximize GPU utilization. The frames are processed in batches to manage memory usage. After processing, results are copied back to the host for video output.



Performance Results and Analysis: We ran the identity transformation with a 128x128 grid size and recorded execution time for varying block dimensions. As the block size increased from 2 to 16, the execution time decreased—from 1.2265 seconds to 0.2143 seconds—indicating improved utilization of the GPU resources. However, increasing the block size further to 32 increased execution time to 0.312251 seconds. This suggests that there is an optimal block size (in this case, 16) where the GPU resources are most efficiently utilized. I expect performance results to align with my expectations up to the optimal block size; beyond that, factors such as increased shared memory usage per block and occupancy limitations may hinder performance.