HPC - 2. Assignment - Parallel Histogram Equalization using CUDA

Kristjan Kostanjšek, Nejc Ločičnik

I. Introduction

For the second assignment, we implemented a Histogram Equalization algorithm, an image processing technique that enhances contrast by redistributing pixel intensities to span a wider range, resulting in a more uniform histogram and improving image clarity. We first developed the sequential version of the algorithm and then compared it to a parallel implementation using CUDA, analyzing their respective speeds. In this report, we will present and analyze the results of each parallelization technique we used.

II. Optimizations and Results

The following subsections detail the optimizations applied to the sequential code, along with the computational speed-up achieved for each optimization. Each optimization is looked at in isolation. This means that the reported optimization speed-ups are for that specific part of the algorithm and not for the full execution. The full execution speed-ups are reported at the end. The speed-up is calculated using the formula: $S = t_S/t_P$, where t_S represents the execution time of the sequential program, and t_P denotes the execution time of the parallel program. All reported times and speed-ups are averages of five runs.

A. RGB to YUV Conversion and Luminance Histogram Computation

The first step in the histogram equalization algorithm involves converting the input RGB image to the YUV color space. This transformation is well-suited for parallel processing since each pixel can be processed independently without dependencies. To optimize performance, we combined the luminance (Y channel) histogram computation within the same kernel function. The histogram calculation leverages CUDA's shared memory, where each block computes a partial histogram of its assigned pixels. These partial histograms are then merged in a final reduction step to produce the complete luminance histogram.

TODO (pls tukaj sam na hitro opisi kaj si ti pol delu se s temi bloki itd...):
- 1D thread scheme (no need for 2D since we aren't working with pixel surroundings, also more efficient thread use for irregular image sizes)
- 2 different approaches with shared memory, 1. every block has a single shared array and we combine them into global memory at the end, 2. every warp (32 threads) has a shared array that then gets combined into a shared block array that then gets combined into global memory

TODO: Result table

TODO: Result comments

B. Cumulative Histogram Computation

TODO: - explain work-efficient optimization - temporary histogram is in shared memory, all 256 threads are in the same block

TODO: Result table

TODO: Result comments

C. New Luminance Computation

This step is straightforward since the values in the luminance histogram are independent of one another. We use 256 threads within the same block to compute each new luminance value in parallel and store the results in a lookup table. This allows for quick assignment of the new values in the next step of the algorithm.

TODO: Result table

TODO: Result comments

D. Assign New Luminance and YUV to RGB Conversion

Following the same efficiency-driven approach as the first step, we combine two operations into a single kernel to maximize efficiency. Both steps are inherently parallelizable since each pixel can be processed independently. First, we assign new luminance values to each pixel using the precomputed lookup table from the previous step. Immediately afterward, we convert the pixels back from YUV to the RGB color space.

TODO: - 2 thread schemes comparison, 1D vs 2D, checking if 2D is any better since pixel locality might use the same values (easier to cache, but the histogram is an array of length 256 and should fit in the cache either way)

TODO: Result table

TODO: Result comments

E. Final results

TODO: - speed ups of full algorithm execution sequential vs. our best performing individual optimizations

TODO: Result table

TODO: Result comments

III. Conclusion

TODO - bom na koncu, lp