Efficient Image Processing: Leveraging CUDA for Real-Time Filter Application and Edit Operations

GROUP MEMBERS -
1- Shreyas Pande
2- Piyush Pancholi
3- Yash Mandawat
4- Niket Srivastav

CS516: Parallelization of Programs

Project Final Phase

VERSIONS OF FILTER EXPLORED

Negative image

It swaps each pixel's color value with its complementary color, creating an image where light areas become dark and vice versa

Blur image

A blur image filter applies a smoothing effect to an image by averaging the colors of neighboring pixels

Mirror image

A mirror image filter reflects the pixels of an image across a central axis, creating a mirrored effect

Square blur image

A square blur filter blurs an image by averaging the color values of each pixel within a square-shaped region around it

Sequential Code

The Sequential function iterates through each pixel in the input image, negates its color values, and assigns them to the corresponding pixel in the output image.

```
for (int y = 0; y < height; ++y) {
    for (int x = 0; x < width; ++x) {
        for (int c = 0; c < channels; ++c) {
            outputImage.at<Vec3b>(y, x)[c] = 255 - inputImage.at<Vec3b>(y, x)[c];
        }
  }
}
```

Iterating through each pixel with the help of nested loop

Negating the value of each pixel.

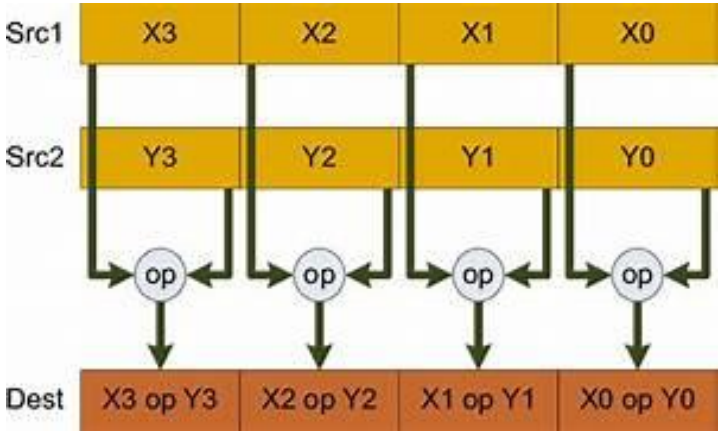
OpenCV Code

Matrices Subtraction: OpenCV allows to perform arithmetic operations directly on objects, which represent images. When you subtract one object from another, OpenCV performs element-wise subtraction.

Efficiency: OpenCV's matrix arithmetic operations are optimized for performance. Under the hood, these operations are often implemented using low-level SIMD instructions or parallel processing techniques to achieve efficient computation

SIMD Instructions

- SIMD instructions are a type of CPU instruction set that enables performing the same operation on multiple data elements simultaneously.
- For example, a SIMD instruction might add four pairs of integers together in one operation, effectively performing the same addition operation on all four pairs concurrently.



CUDA Code

The CUDA kernel that will be executed on the GPU. It calculates the transformation for each pixel in parallel. Each thread in the GPU grid corresponds to one pixel in the image

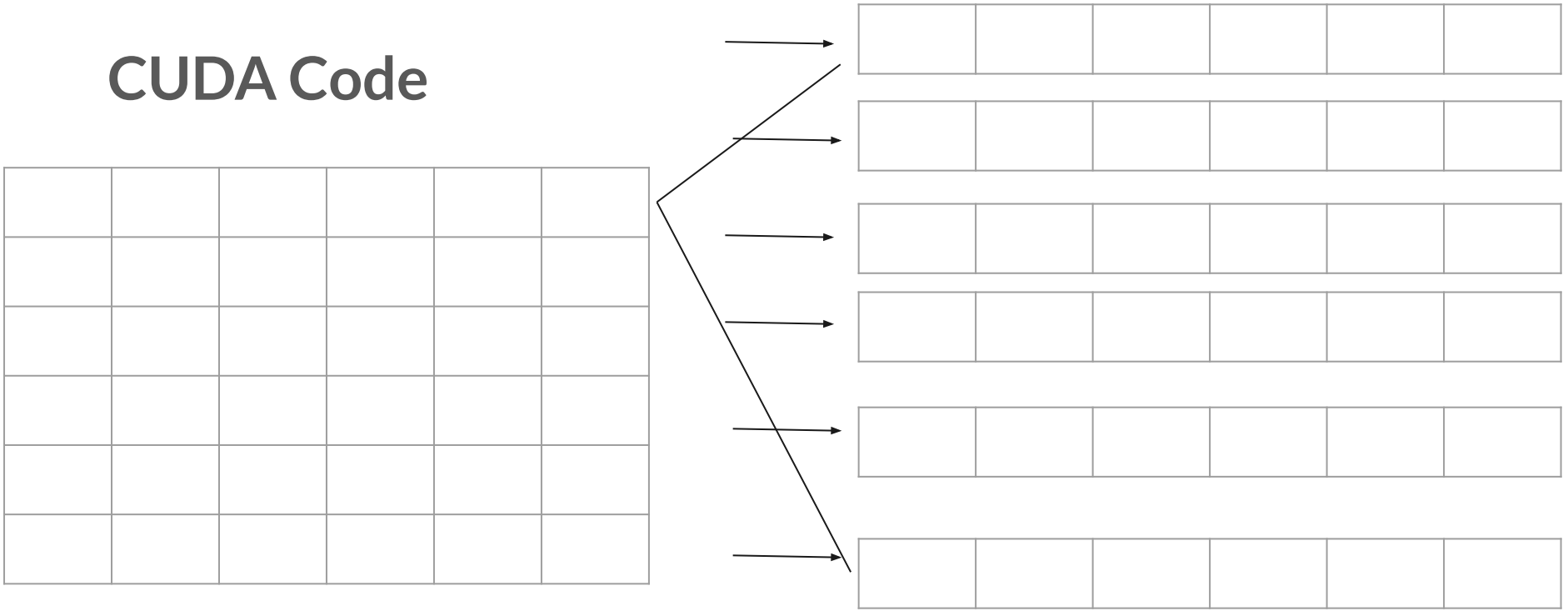
```
__global__ void negativeImageKernel(unsigned char* input,
 unsigned char* output, int width, int height, int channels) {
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;

    if (x < width && y < height) {
        int idx = (y * width + x) * channels;
        for (int c = 0; c < channels; ++c) {
            output[idx + c] = 255 - input[idx + c];
        }
    }
}
```

Negative Blur Filter

CUDA Code

Applying Operation parallelly on rows



Blocks and Threads

- The number of blocks along the x-axis (gridDim.x) is calculated as (width + blockDim.x - 1) / blockDim.x.
- The number of blocks along the y-axis (gridDim.y) is calculated as (height + blockDim.y - 1) / blockDim.y.
- The number of threads per block is specified by the blockDim variable in the code.
- **blockDim.x = 32** and **blockDim.y = 32**, each block contains a total of **1024 threads**.
- Threads within a block are organized in a two-dimensional grid fashion, with dimensions specified by blockDim.x and blockDim.y.

WHY 32 Blocks and 1024 Threads

- Choosing a larger block size can increase the occupancy of the GPU, which can lead to better utilization of GPU resources and potentially higher performance.
- However, larger block sizes also increase register and shared memory usage, which may limit the number of blocks that can run concurrently on the GPU.
- A block size of 32x32 (1024 threads) is often chosen because it allows for efficient memory access patterns and optimal utilization of GPU resources.

Negative image

A negative image filter, also known as an invert filter, transforms an image by inverting the colors of each pixel. In other words, it converts bright areas to dark and vice versa, creating a photographic negative effect.

Here's how it typically works:

1. For each pixel in the image, calculate the inverse of its color value.
2. Replace the original color value with its inverse.

Image preview

Blur image

A blur image filter, is a technique used to reduce the level of detail in an image, resulting in a smoother appearance.

There are various methods for blurring images, but one of the most common is the Gaussian blur.

Here's how Gaussian blur typically works:

1. Define a Gaussian kernel, which is essentially a matrix of numbers that represents a bell-shaped curve.
2. Place the kernel over each pixel in the image.
3. Multiply each pixel in the kernel by the corresponding pixel in the image.
4. Sum up the results to obtain the new value for the central pixel.
5. Repeat this process for every pixel in the image.

Image preview

Mirror Image Filter

- A mirror image filter, commonly used in image processing, flips an image horizontally, creating a mirrored version of the original.
- This filter essentially reflects the pixels of the image across a vertical axis, resulting in a symmetrical effect.

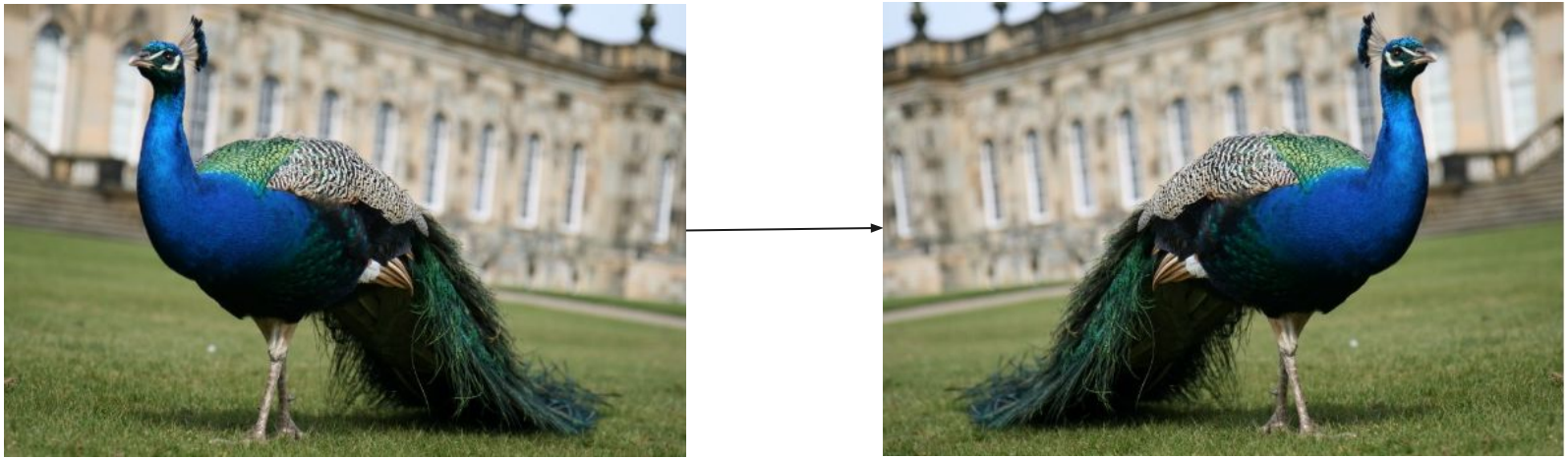
127	255	220	112	255
110	90	0	190	110
0	127	100	255	120
27	45	91	127	0
255	200	190	185	140

Original Image

255	112	220	255	127
110	190	0	90	110
120	255	100	127	0
0	127	91	45	27
140	185	190	200	255

Mirror Image

Image preview



Square Blur Filter

1. Define a square-shaped kernel of a specified size (for example, 3x3, 5x5, etc.).
2. Place the kernel over each pixel in the image.
3. Calculate the average color value of all the pixels within the kernel.
4. Replace the color value of the current pixel with the calculated average.

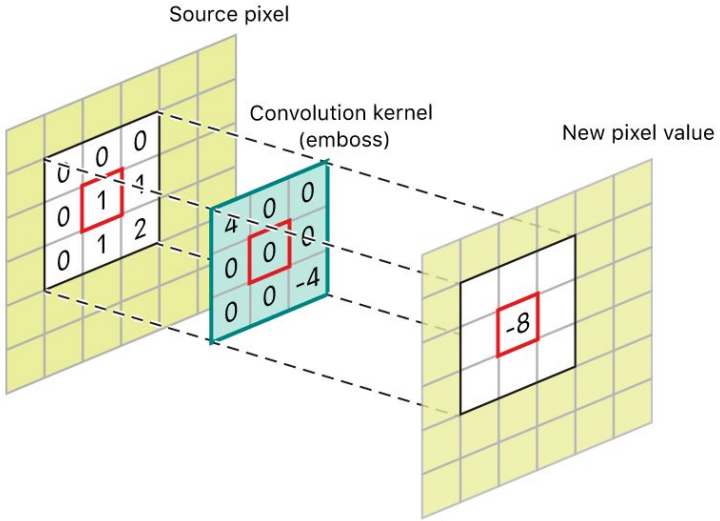


Image preview

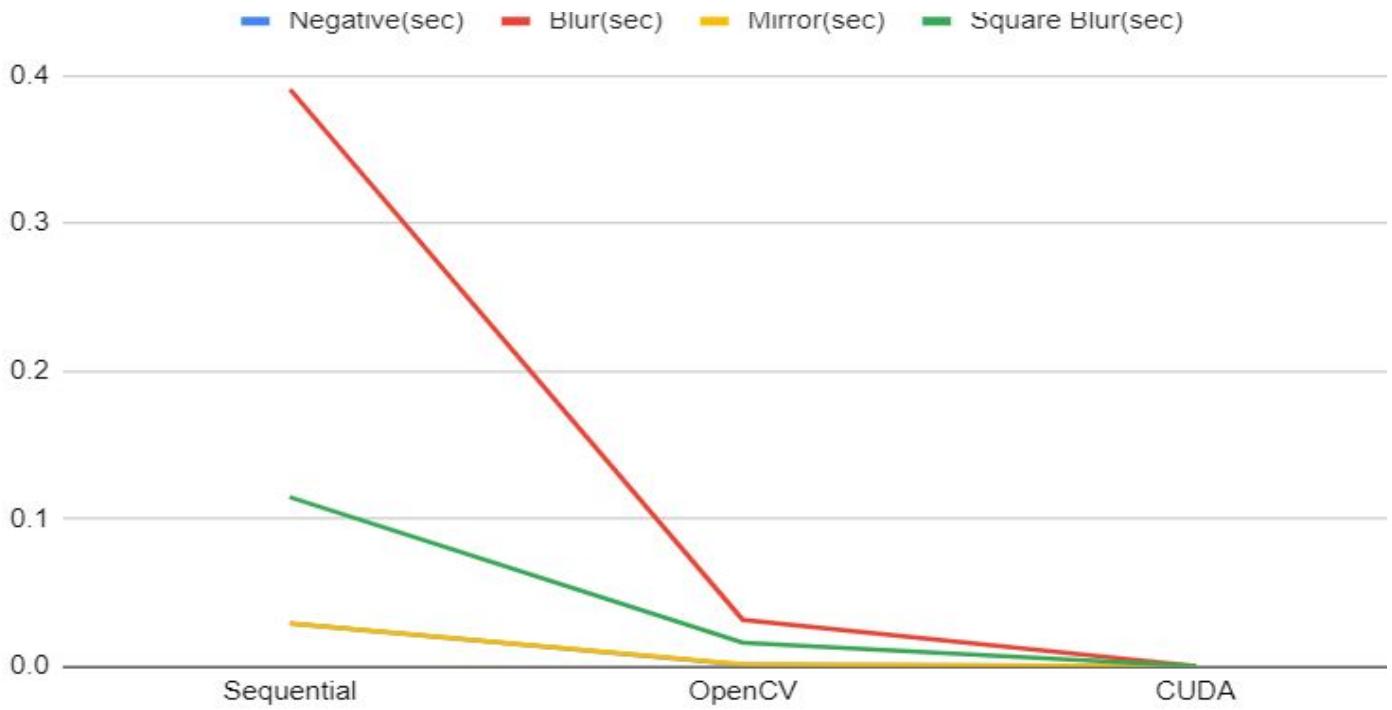
PERFORMANCE MATRIX (IMAGE SIZE 850 X 500)

FILTER / CODE	Negative(sec)	Blur(sec)	Mirror(sec)	Square Blur(sec)
Sequential	0.029154	0.390594	0.029293	0.114616
OpenCV	0.001181	0.031415	0.001503	0.016189
CUDA	0.000177	0.000215	0.000194	0.000247

PERFORMANCE MATRIX (IMAGE SIZE 5000X3000)

FILTER / CODE	Negative(sec)	Blur(sec)	Mirror(sec)	Square Blur(sec)
Sequential	0.684119	16.015906	0.701823	4.241223
OpenCV	0.038412	0.156188	0.064382	0.223763
CUDA	0.124507	0.057410	0.042904	0.043987

Graph


Negative(sec)
Blur(sec)
Mirror(sec)
Square Blur(sec)
0.4
0.3
0.2
0.1
0.0
Sequential
OpenCV
CUDA

HardWare Platform

Our experiments will primarily utilize **NVIDIA CUDA- enabled GPU Tesla T4 GPU** present in Google Collab as of now to harness the parallel processing capabilities efficiently. Specifically, we will employ GPU with varying specifications to evaluate performance across different hardware configurations and capabilities. The Gpu would be having a total of **40 streaming processors and a maximum thread block size of 1024. The overall memory size available in each SM is : 64Kb.**

Thank You!!