IMPLEMENTING AND COMPARING CONVOLUTION, FILTERING AND MORPHOLOGICAL OPERATIONS FOR IMPROVEMENT IN PERFORMANCE WITH GPU ACCELERATION USING GAUSSIAN BLUR, MEAN FILTERING AND DILATION

Dhruv Bajaj
Satwik Saurav
Somya Sahu

PROBLEM STATEMENT

To evaluate the efficacy of CUDA C for accelerating image processing algorithms compared to traditional serial CPU implementations.

METHODOLOGY

ENVIRONMENT SETUP:

- Utilized google colab with a google T4 processor for GPU acceleration.
- Converted serial python code to a c binary using pyinstaller.
- Developed parallel cuda c code for efficient execution on the t4 processor.

Benchmarking process:

- Selected test images of varying resolutions.
- Executed serial and parallel versions of image processing algorithms (gaussian blur, dilation, and mean filtering).
- Ran each implementation multiple times (10 iterations) for accuracy.

RUNTIME MEASUREMENT AND CALCULATION:

- ❑ Recorded start and end times using high-resolution timers.
- ❑ Calculated average execution times for each implementation and image combination.
- ❑ Quantified performance improvement using the speedup factor.

Data analysis and reporting:

- ❑ Compiled and analyzed data, including average execution times and speedup factors.
- ❑ Presented results visually with charts and graphs.
- ❑ Discussed findings, emphasizing the impact of algorithms, image sizes, and formats on execution times, along with limitations and suggestions for further exploration.

INFRASTRUCTURE

The serial code:

- ❑ Made in python compiled to C binaries using pyinstaller
- ❑ Run on T4 processor in google colab
- ❑ Implemented in serial fashion

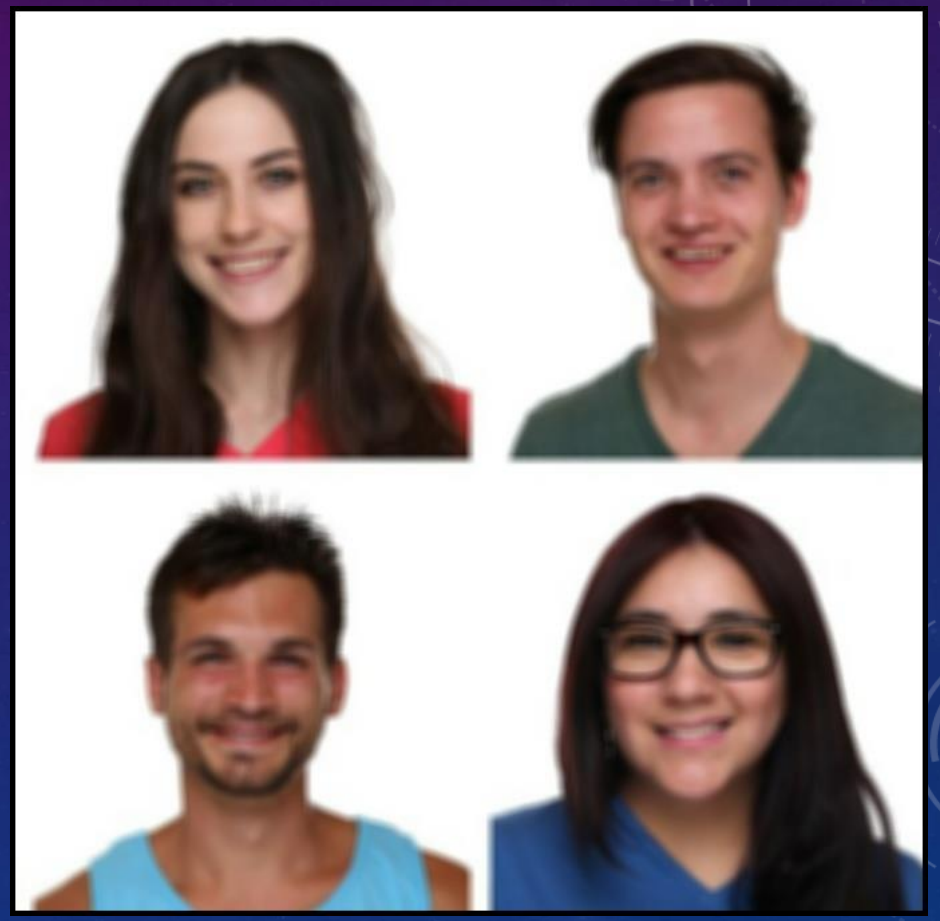
The parallel code:

- ❑ Made in CUDA C
- ❑ Run on T4 processor in google colab
- ❑ Implemented using parallel computing techniques

CUDA IMPLEMENTATION OF MEAN FILTER

```
#include "opencv2/imgproc/imgproc.hpp"
#include <opencv2/highgui.hpp>
#include <iostream>
#include <string>
#include <stdio.h>
#include <cuda.h>
#include "cuda_runtime.h"

#define BLOCK_SIZE      16
#define FILTER_WIDTH    8
#define FILTER_HEIGHT   8

using namespace std;

// Run Mean Filter on GPU
__global__ void meanFilter(unsigned char *srcImage, unsigned char *dstImage, unsigned int width,
unsigned int height, int channel)
{
   int x = blockIdx.x*blockDim.x + threadIdx.x;
   int y = blockIdx.y*blockDim.y + threadIdx.y;

   // only threads inside image will write results
   if((x>=FILTER_WIDTH/2) && (x<(width-FILTER_WIDTH/2)) && (y>=FILTER_HEIGHT/2) && (y<(height-
FILTER_HEIGHT/2)))
   {
      for(int c=0 ; c<channel ; c++)
      {
         unsigned char filterVector[FILTER_WIDTH*FILTER_HEIGHT];
         int sum = 0;
         // Loop inside the filter to sum pixel values
         for(int ky=0; ky<FILTER_HEIGHT; ky++) {
            for(int kx=0; kx<FILTER_WIDTH; kx++) {
               filterVector[ky*FILTER_WIDTH+kx] = srcImage[((y+ky-FILTER_HEIGHT/2)*width + (x+kx-
FILTER_WIDTH/2))*channel+c];
               sum += filterVector[ky*FILTER_WIDTH+kx];
            }
    }

         // Calculate mean value
         dstImage[(y*width+x)*channel+c] = sum / (FILTER_WIDTH*FILTER_HEIGHT);
      }
   }
}
```

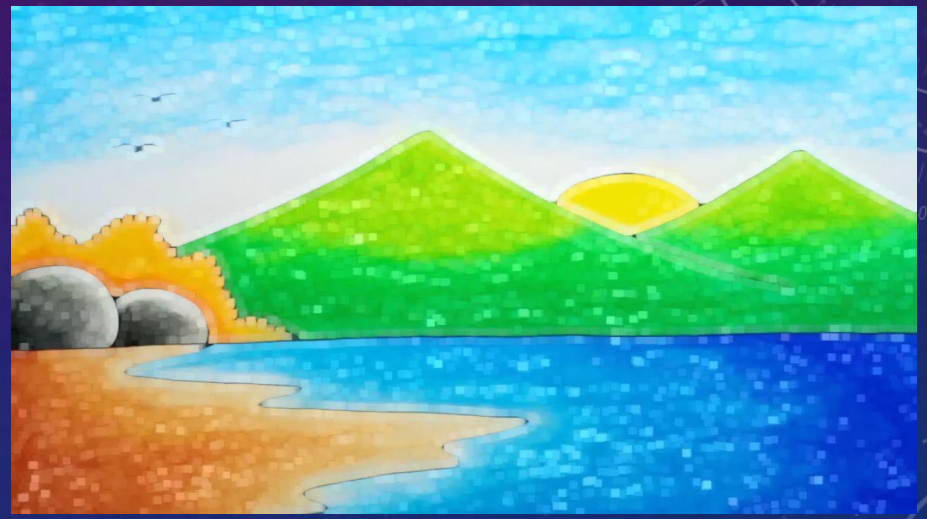
ORIGINAL VS RESULT OF MEAN FILTERING:



CUDA IMPLEMENTATION OF DILATIONFILTER

```
__global__ void dilationFilter(unsigned char *srcImage, unsigned char *dstImage, unsigned int width,
unsigned int height, int channel)
{
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;

    // Check if the thread is within the image boundaries
    if (x < width && y < height)
    {
        for (int c = 0; c < channel; c++)
        {
            unsigned char maxPixel = 0; // Initialize maxPixel to the minimum possible value

            // Loop through the filter window
            for (int ky = -FILTER_HEIGHT / 2; ky <= FILTER_HEIGHT / 2; ky++)
            {
                for (int kx = -FILTER_WIDTH / 2; kx <= FILTER_WIDTH / 2; kx++)
                {
                    // Calculate the neighbor's coordinates
                    int nx = x + kx;
                    int ny = y + ky;

                    // Check if the neighbor is within the image boundaries
                    if (nx >= 0 && nx < width && ny >= 0 && ny < height)
                    {
                        // Get the neighbor's pixel value
                        unsigned char neighborPixel = srcImage[(ny * width + nx) * channel + c];

                        // Update maxPixel if the neighbor's pixel value is greater
                        if (neighborPixel > maxPixel)
                        {
                            maxPixel = neighborPixel;
                        }
                    }
                }
            }

            // Set the dilation result
            dstImage[(y * width + x) * channel + c] = maxPixel;
        }
    }
}
```

ORIGINAL VS RESULT DILATION:

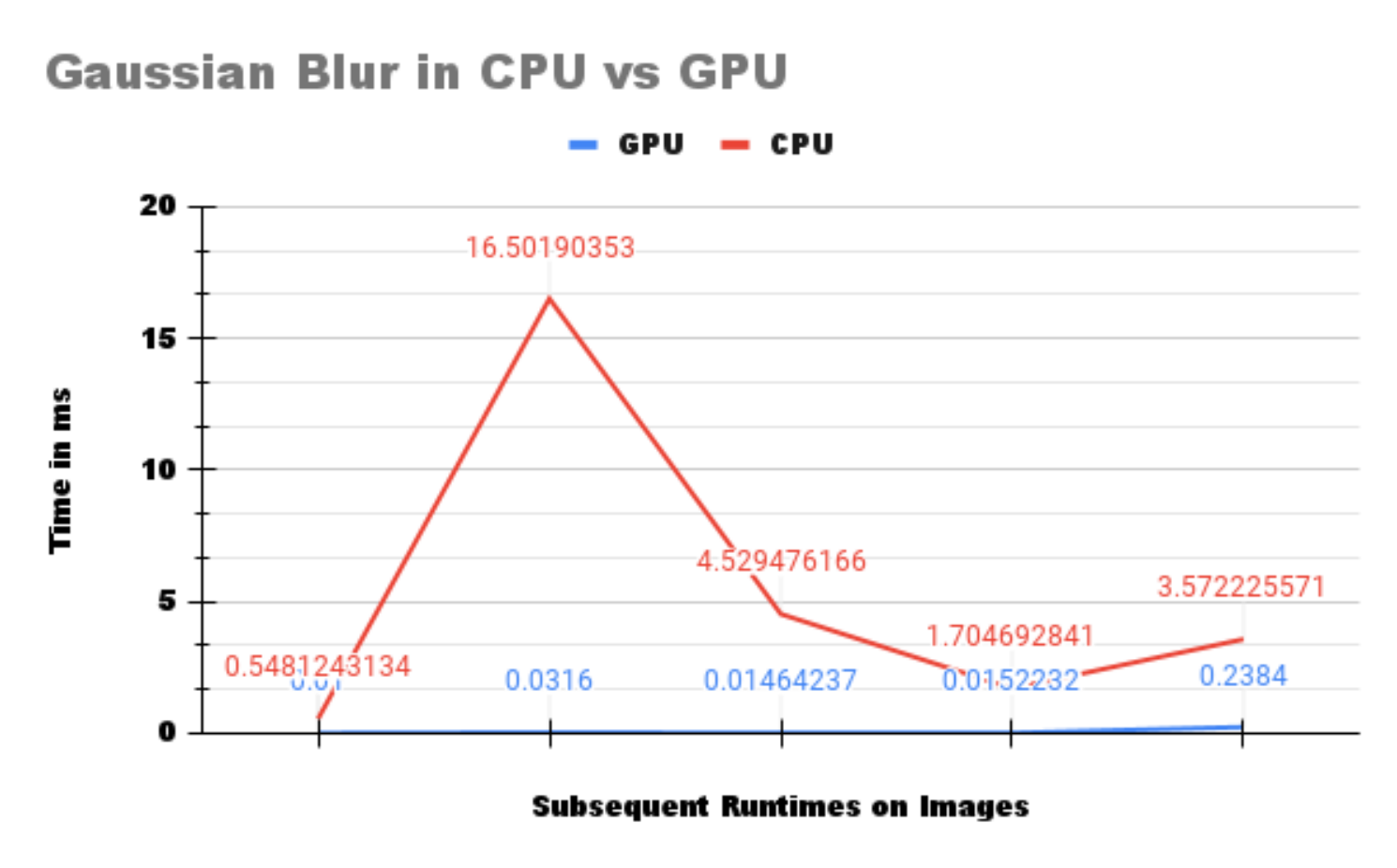


CUDA IMPLEMENTATION OF GUSSIANFILTER

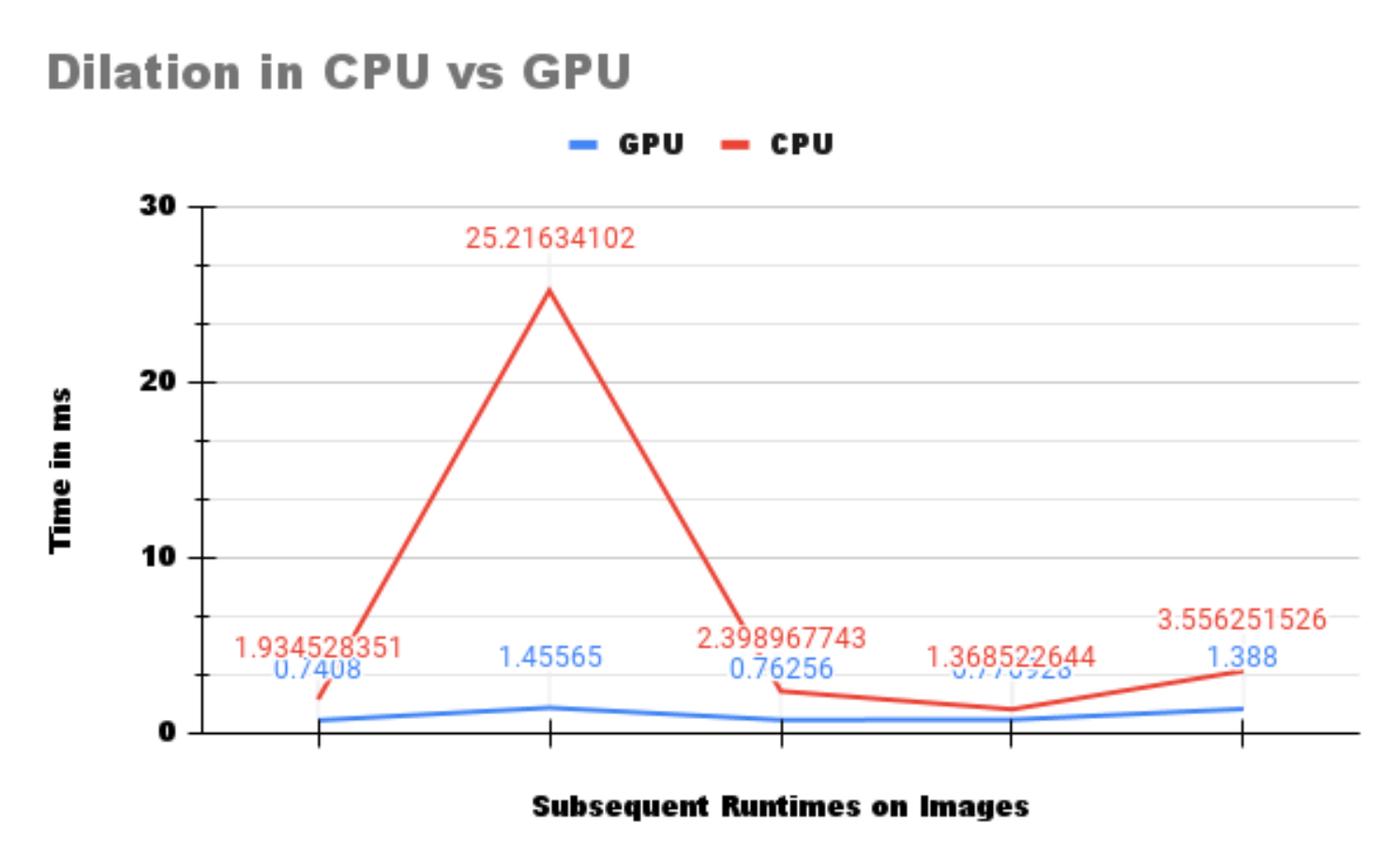
```
__global__ void gaussianFilter(unsigned char *srcImage, unsigned char *dstImage, unsigned int width,
unsigned int height, int channel, float sigma) {
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;

    __shared__ float gaussianKernel[FILTER_WIDTH * FILTER_HEIGHT];

    if (threadIdx.x < FILTER_WIDTH && threadIdx.y < FILTER_HEIGHT) {
        int index = threadIdx.y * FILTER_WIDTH + threadIdx.x;
        gaussianKernel[index] = exp(-(threadIdx.x * threadIdx.x + threadIdx.y * threadIdx.y) / (2 *
sigma * sigma)) / (2 * M_PI * sigma * sigma);
    }

    __syncthreads();

    if (x >= FILTER_WIDTH / 2 && x < width - FILTER_WIDTH / 2 && y >= FILTER_HEIGHT / 2 && y < height -
FILTER_HEIGHT / 2) {
        for (int c = 0; c < channel; c++) {
            float sum = 0.0f;
            int index = 0;
            for (int ky = -FILTER_HEIGHT / 2; ky <= FILTER_HEIGHT / 2; ky++) {
                for (int kx = -FILTER_WIDTH / 2; kx <= FILTER_WIDTH / 2; kx++) {
                    int srcX = x + kx;
                    int srcY = y + ky;
                    sum += srcImage[(srcY * width + srcX) * channel + c] * gaussianKernel[index];
                    index++;
                }
            }
            dstImage[(y * width + x) * channel + c] = sum;
        }
    }
}
```

ORIGINAL VS RESULT GAUSSIAN BLUR:

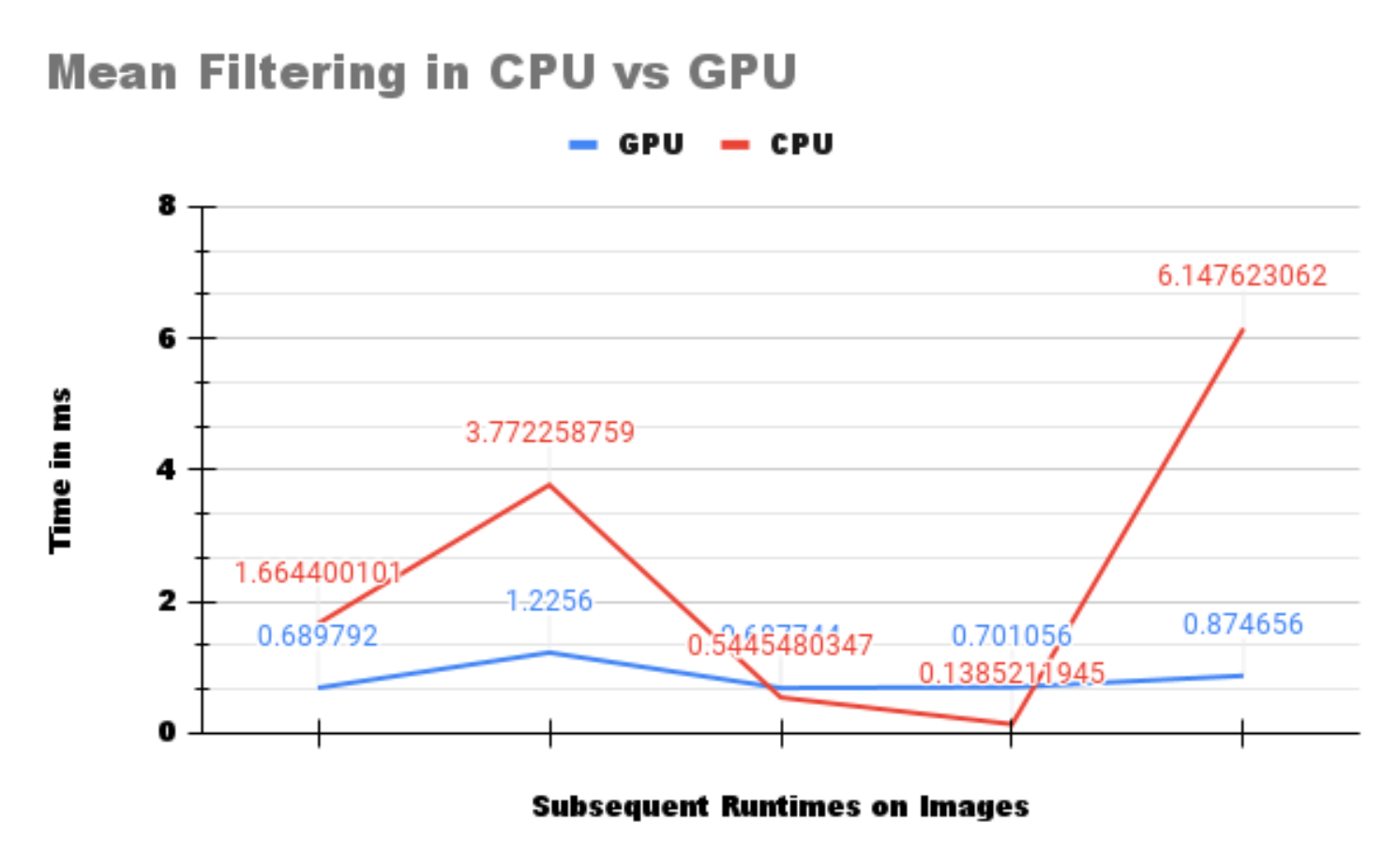
RESULTS


Gaussian Blur in CPU vs GPU
GPU
CPU
20
15
10
5
0
Time in ms
16.50190353
4.529476166
3.572225571
1.704692841
0.5481243134
0.0316
0.01464237
0.0152232
0.2384
Subsequent Runtimes on Images

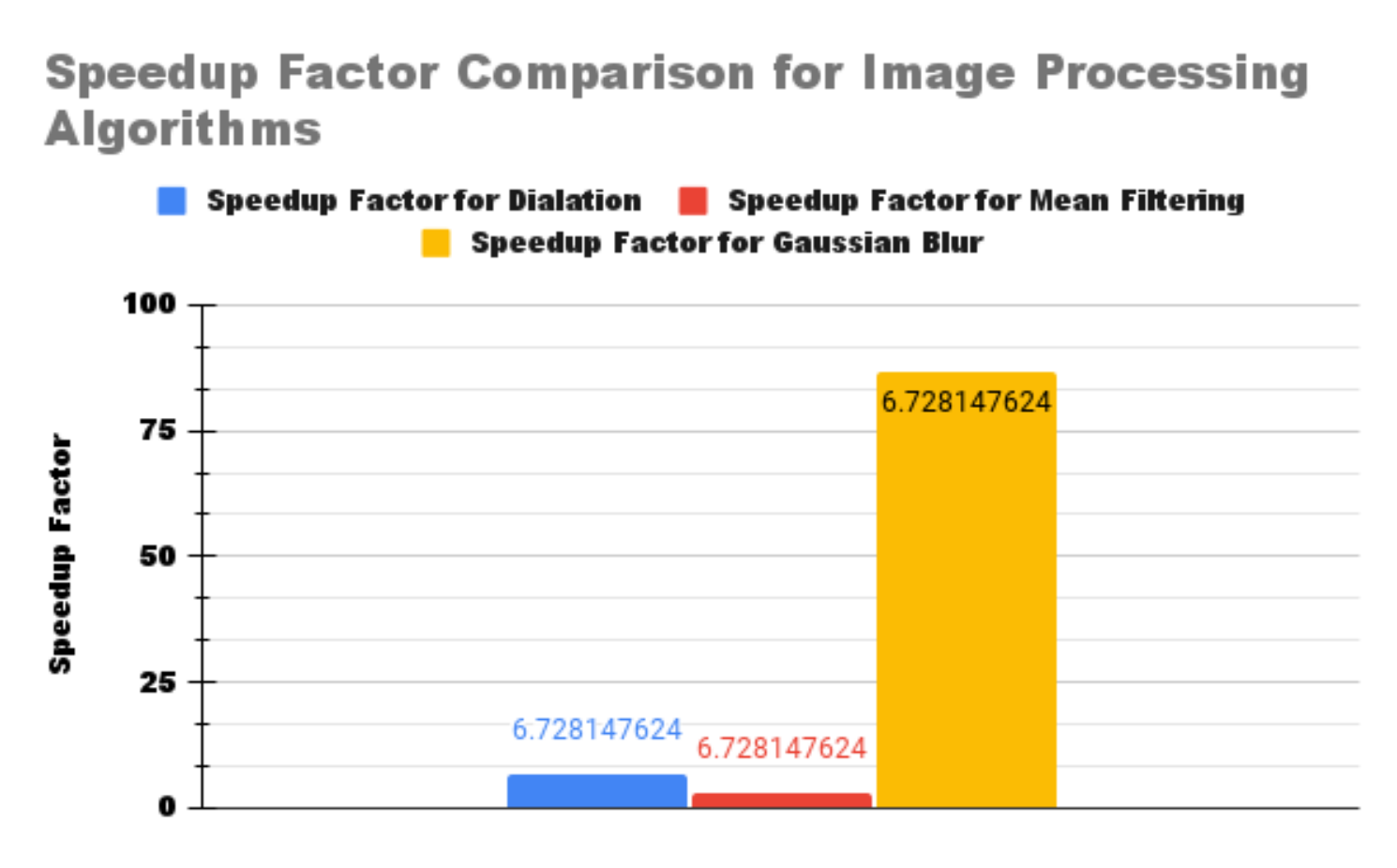
RESULTS


Dilation in CPU vs GPU
GPU
CPU
30
20
10
0
Time in ms
25.21634102
1.934528351
0.7408
1.45565
2.398967743
0.76256
1.368522644
3.556251526
1.388
Subsequent Runtimes on Images

RESULTS


Mean Filtering in CPU vs GPU
GPU
CPU
Time in ms
8
6
4
2
0
6.147623062
3.772258759
1.664400101
1.2256
0.689792
0.5445480347
0.701056
0.874656
0.138521945
Subsequent Runtimes on Images

RESULTS


Speedup Factor Comparison for Image Processing Algorithms
Speedup Factor for Dialation
Speedup Factor for Mean Filtering
Speedup Factor for Gaussian Blur
Speedup Factor
100
75
50
25
0
6.728147624
6.728147624
6.728147624

REFERENCES

- Xi Chen, Yuehong Qiu, and Hongwei Yi. Implementation and performance of image filtering on gpu. In 2013 Fourth International Conference on Intelligent Control and Information Processing (ICICIP), pages 514–517, 2013.
- Yonny S. Izmantok. Time complexity measurement on cuda-based gpuparallel architecture of morphology operation. Korean Journal of Advanced Science and Technology, page 9, 2013.
- Mugdha A. Rane. Fast morphological image processing on gpu using cuda. page 37, 2023.
- Ferhat Bozkurt. Effective gaussian blurring process on graphics processing unit with cuda. International Journal of Machine Learning, 5(483):W012, 2023.
- Mete Ya˘gano˘glu Ferhat Bozkurt and Faruk Baturalp Gunay. Effective gaussian blurring process on graphics processing unit with cuda. International Journal of Machine Learning, (483):1–5, 2024.
- Mouna Afif. Efficient 2d convolution filters implementations on graphics processing unit using nvidia cuda. International Journal of Computer Science and Mobile Computing, 6(10):1–5,2020.
- Munesh Chauhan. Optimizing gaussian blur filter using cuda parallel framework. International Journal of Computer Science and Mobile Computing, 6(10):1–5, 2018.
- Nahla M. Ibrahim. Gaussian blur through parallel computing. page 5.
- Department of Computer Science and Applications, Gandhigram Rural Institute –Deemed University Gandhigram,page 13, 2016.