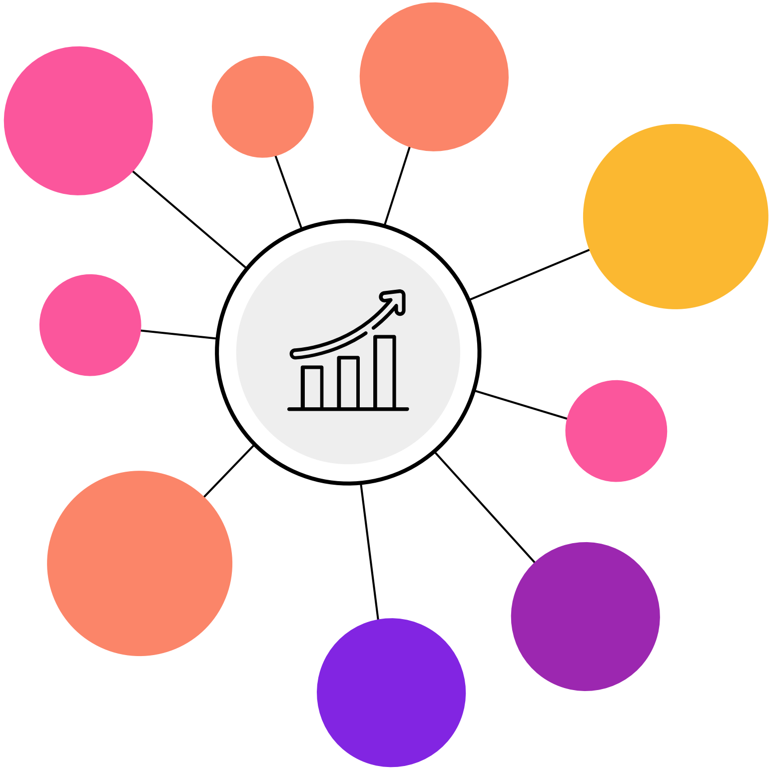


Image Enhancement In Frequency Domain Using Gaussian Filter

LIM YU HER 23WMR09454
LOH XIN JIE 23WMR10840

Project Objective

1

Improve image quality

Through Unsharp Masking

2

Improve performance

Through applying parallel computing

Platforms Used

Open MP

C++, run using Visual Studio 2022, for parallel computing

CUDA

C++, run using Visual Studio 2022, for parallel computing

Seaborn, Matplotlib

Python, run using script and ipykernel

General process flow

Convert RGB to Greyscale

Simplify our problem complexity

Zero pad image

Dimensions required to be powers of 2.

FFT

Cooley Tukey 1D FFT.

Unsharp Masking

With **Gaussian Highpass Filter.**

IFFT

Obtain enhanced image.

Unpad image

Obtain original dimensions of image.

Areas to be parallelized

FFT

Reason

Possible to be broken down into smaller independent sub problems.

Gaussian Highpass Filter

Reason

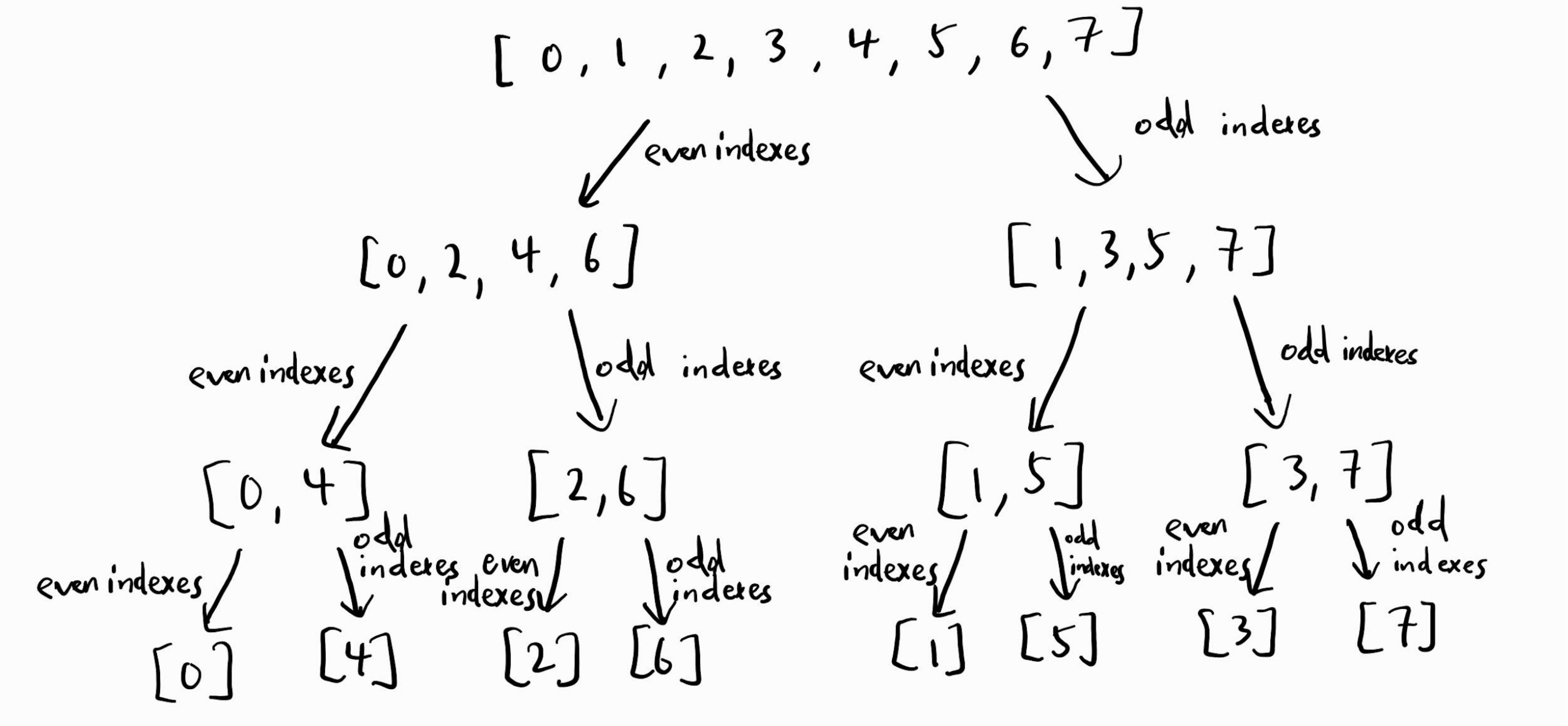
Compute and apply filter are both Independent operations

IFFT

Reason

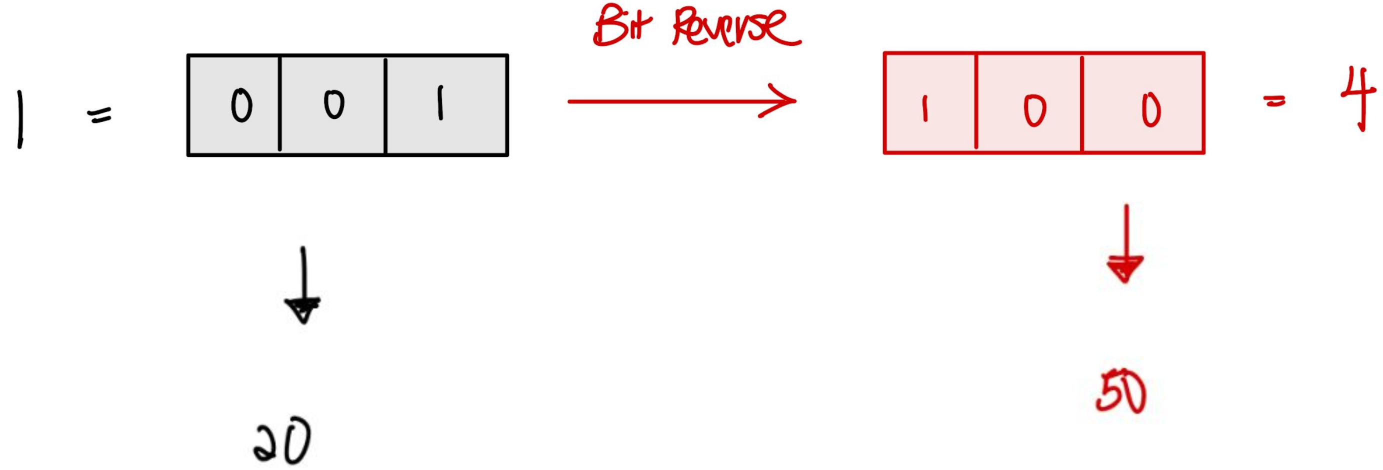
Same reason as FFT

Recursive 1D FFT using Cooley Tukey Algorithm (splitting step)



Iterative 1D FFT using Cooley Tukey Algorithm

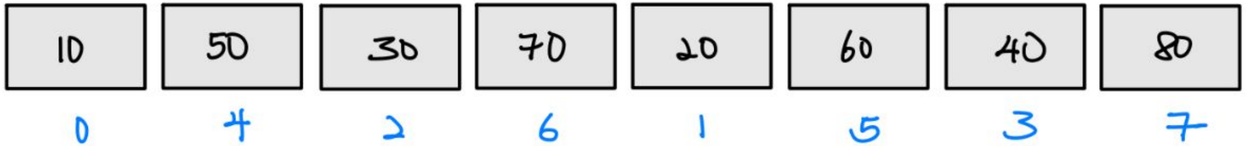
Iterative FFT using Cooley Tukey Algorithm



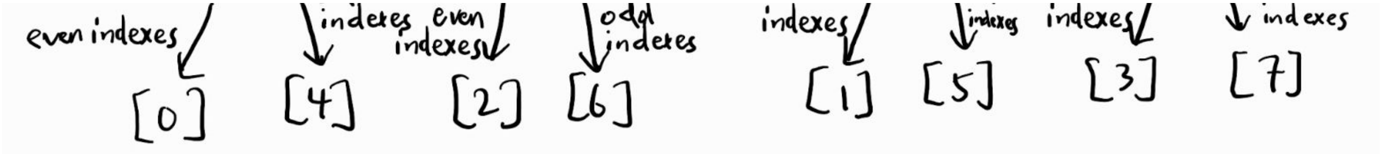
Iterative FFT using Cooley Tukey Algorithm

Iterative & Recursive Splitting Final Result

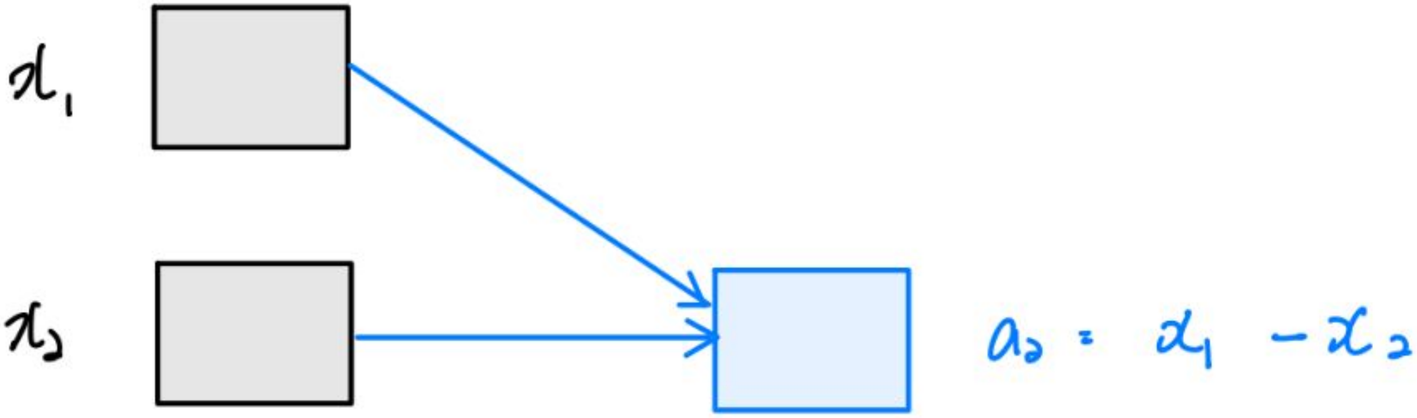
Iterative Final Result :



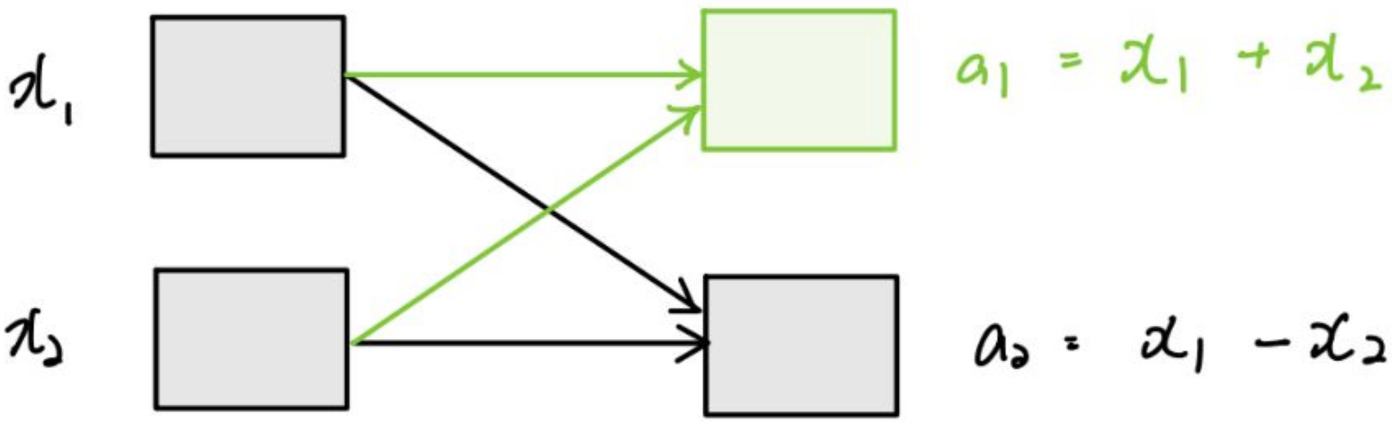
Recursive Final Result :



Combination Step : Twiddle Factor



Combination Step : Twiddle Factor

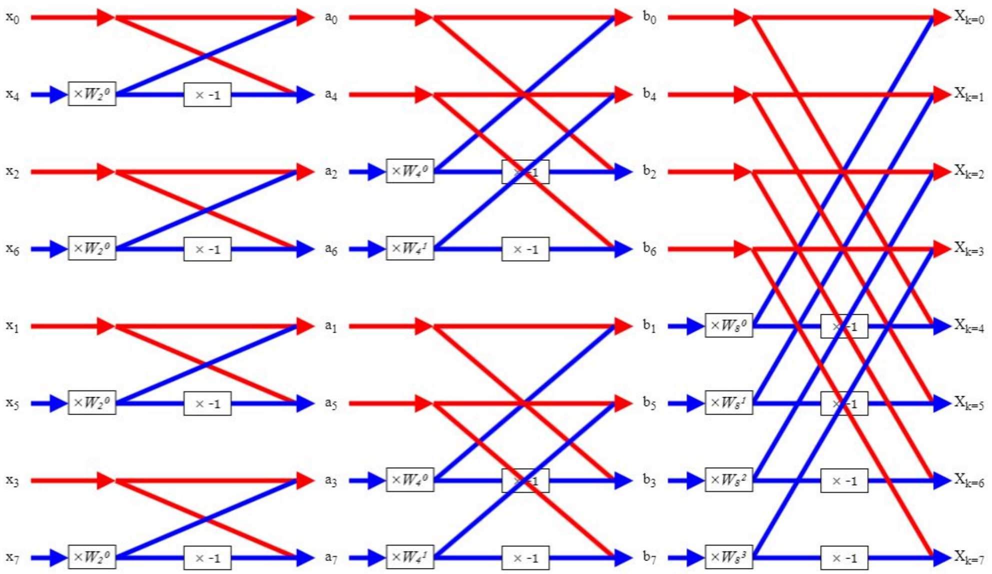


Combination Step : Twiddle Factor

$$W_N^k = e^{\frac{-i 2\pi k}{N}}$$

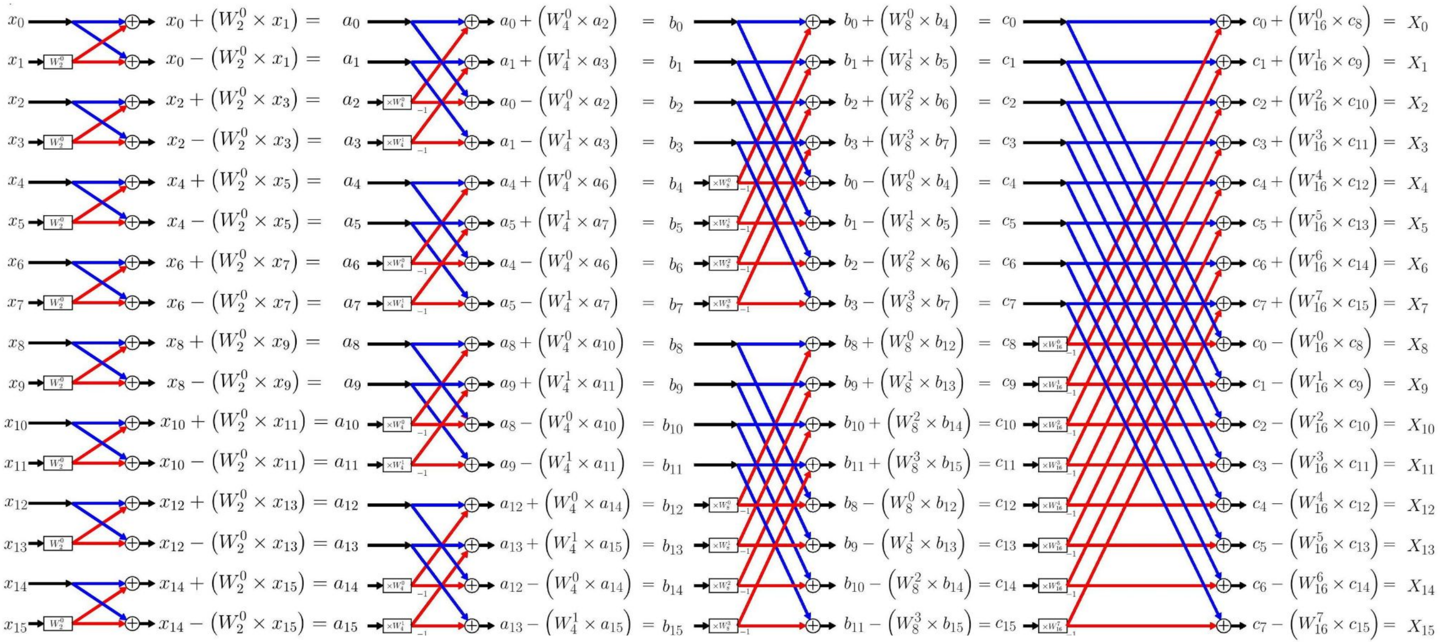
$$W_N^k = \cos\left(2\pi \times \frac{k}{N}\right) - i \sin\left(2\pi \times \frac{k}{N}\right)$$

WHY Parallel FFT



$$8 \times \log_2(8) = 24$$

WHY Parallel FFT



$16 \times \log_2(16) = 64$

Gaussian High Pass Filter

Fig. 3: original image

Fig. 5: Gaussian high pass filter

The Gaussian high pass filter is given as:

$$H(u,v) = 1 - e^{-D^2(u,v)/2{D_0}^2}$$

The formula for applying a **Gaussian High-Pass Filter** is:

$$G_{\text{highpass}}(u,v) = F(u,v) \cdot H(u,v)$$

The cutoff frequency (D_0) controls which frequencies are filtered: lower D_0 blurs more, while higher D_0 sharpens details.

Unsharp Masking using Gaussian Highpass Filter

Original Image

Unsharp Masking

$$G_{\text{sharpened}}(x, y) = G(x, y) + \alpha \cdot G_{\text{highpass}}(x, y)$$

α: The scaling factor controlling the intensity of sharpening ($\alpha > 0$).

The Different Between FFT & IFFT

$$F(u,v) = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} A(x,y) e^{\boxed{-j} 2\pi\left(\frac{ux}{N} + \frac{vy}{M}\right)}$$

$$A(x,y) = \boxed{\frac{1}{NM}} \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} F(u,v) e^{\boxed{j} 2\pi\left(\frac{ux}{N} + \frac{vy}{M}\right)}$$

WHY Normalize

Original

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

After FFT

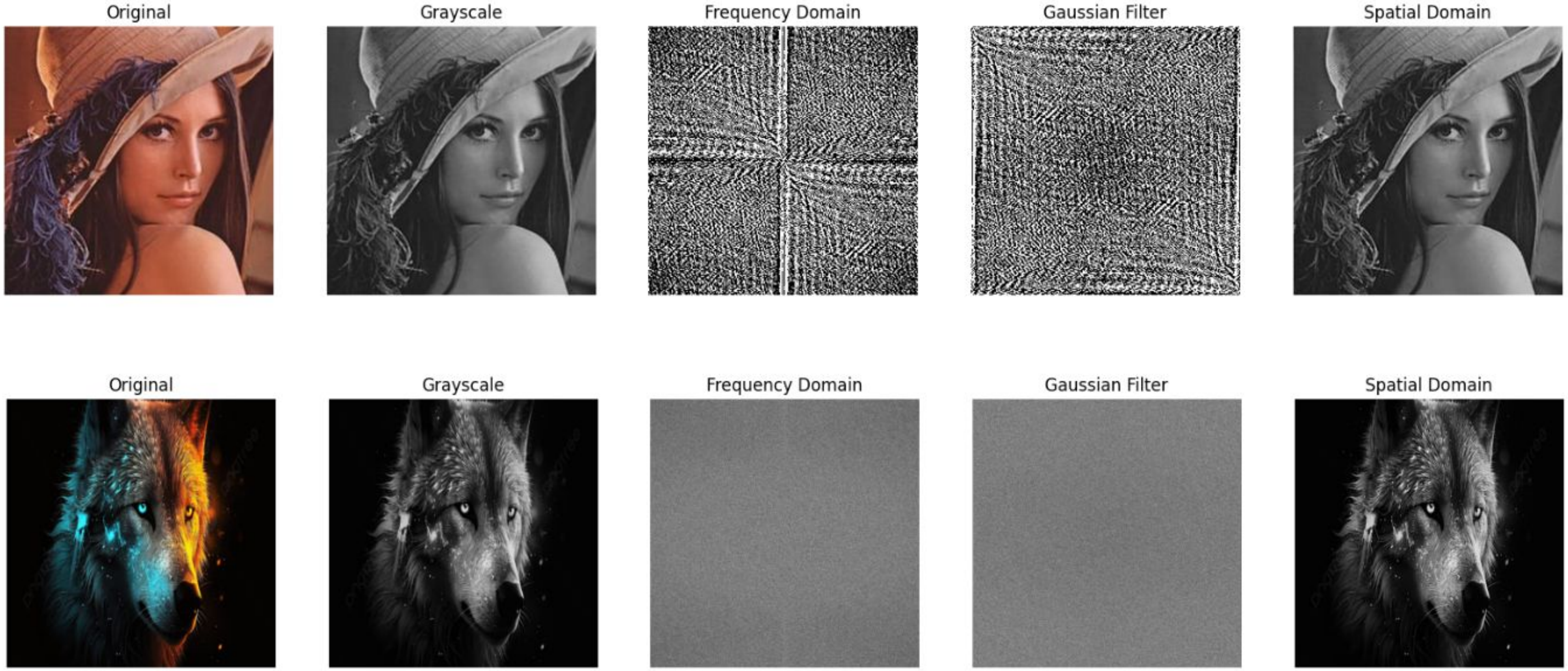
$$\begin{bmatrix} 136+0j & -8+8j & -8+0j & -8-8j \\ -32+32j & 0+0j & 0+0j & 0+0j \\ -32+0j & 0+0j & 0+0j & 0+0j \\ -32-32j & 0+0j & 0+0j & 0+0j \end{bmatrix}$$

WHY Normalize

After IFFT

$$\begin{bmatrix} 16+0j & 32+0j & 48+0j & 64+0j \\ 80+0j & 96+0j & 112+0j & 128+0j \\ 144+0j & 160+0j & 176+0j & 192+0j \\ 208+0j & 224+0j & 240+0j & 256+0j \end{bmatrix}$$

Result - Grayscale



Result - Grayscale (Before & After)

Before

After

Result - Grayscale (Before & After)

Before

After

Result - Grayscale (Before & After)

Original

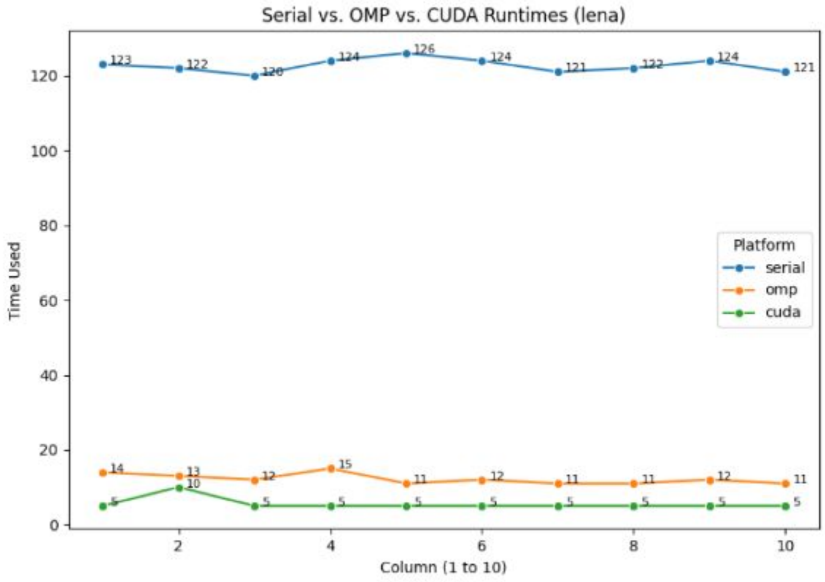
Applied once

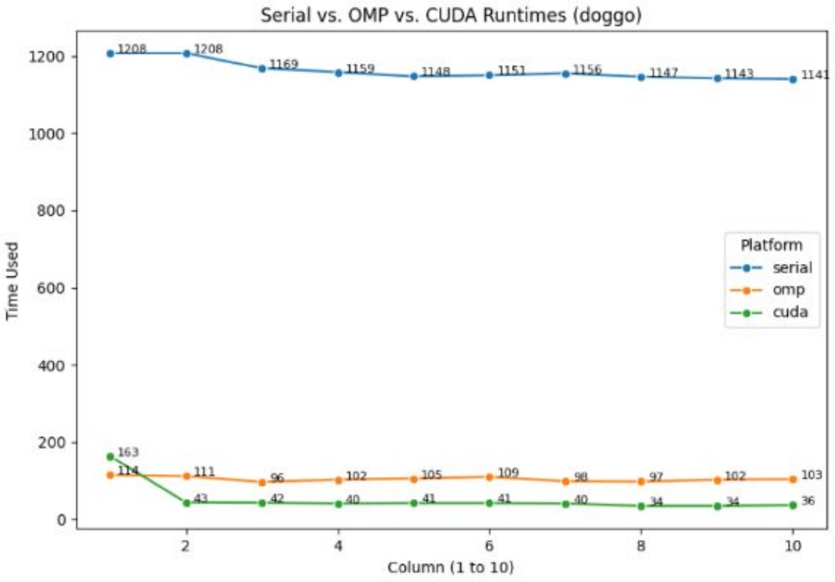
Result - Grayscale (Before & After)

Original

Applied four times

Result - Grayscale (Runtimes)

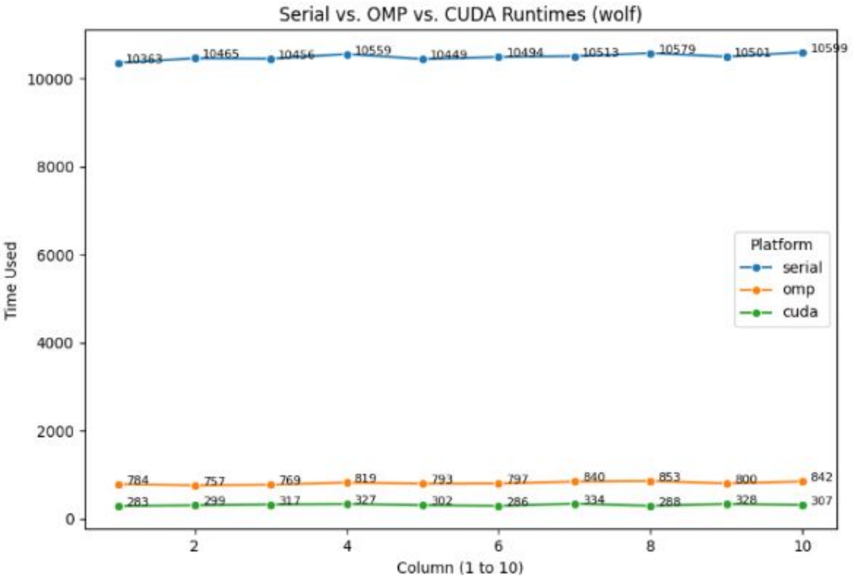


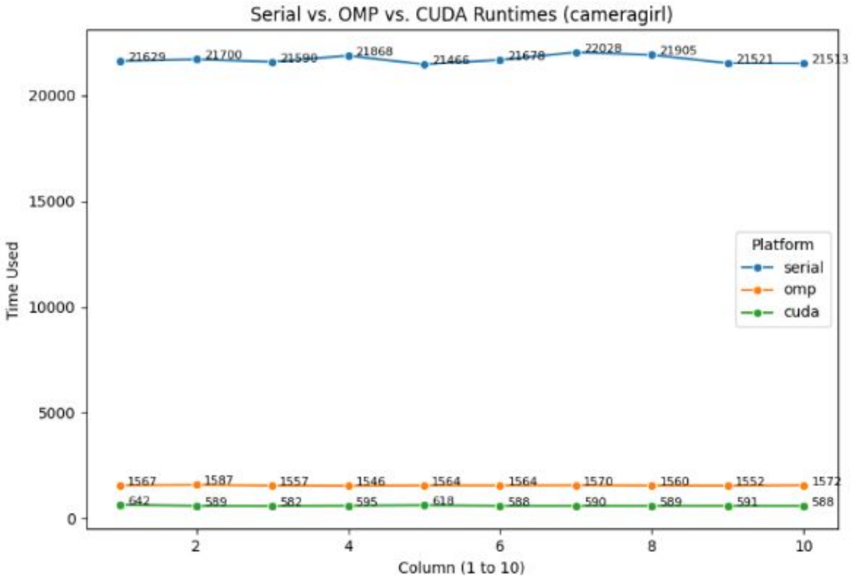


Threads used for OMP: 16

Threads per block used for CUDA: 256

Result - Grayscale (Runtimes)



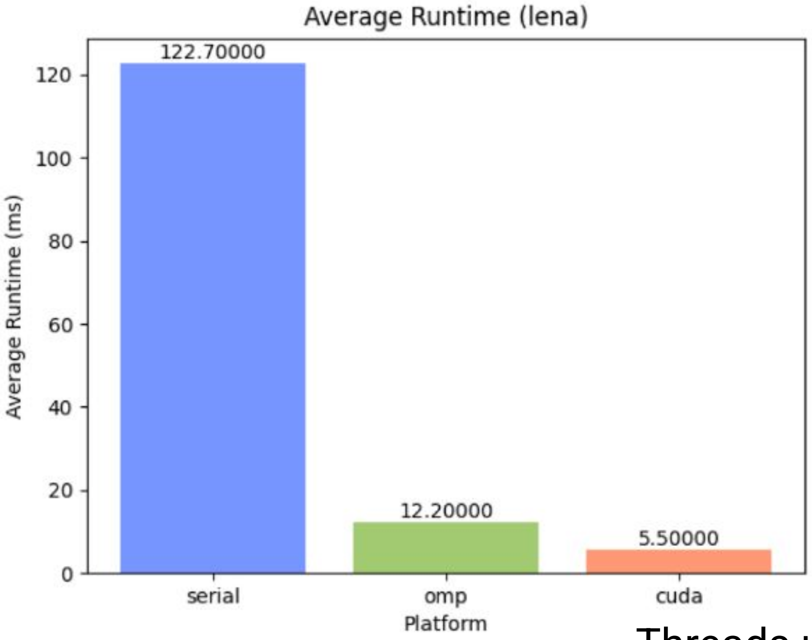


Threads used for OMP: 16

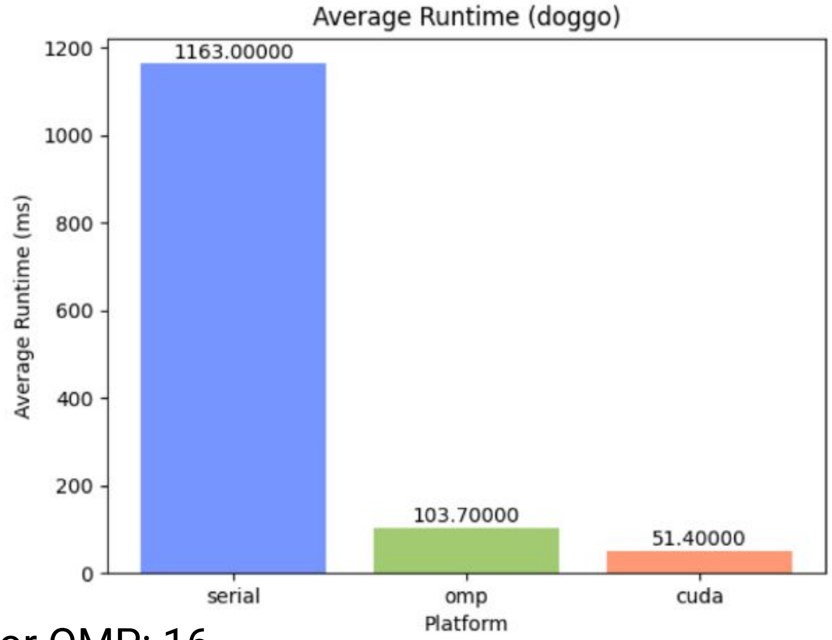
Threads per block used for CUDA: 256

Result - Grayscale (Average Runtimes)

Dimensions : 256 x 256, Size : 28.6 KB



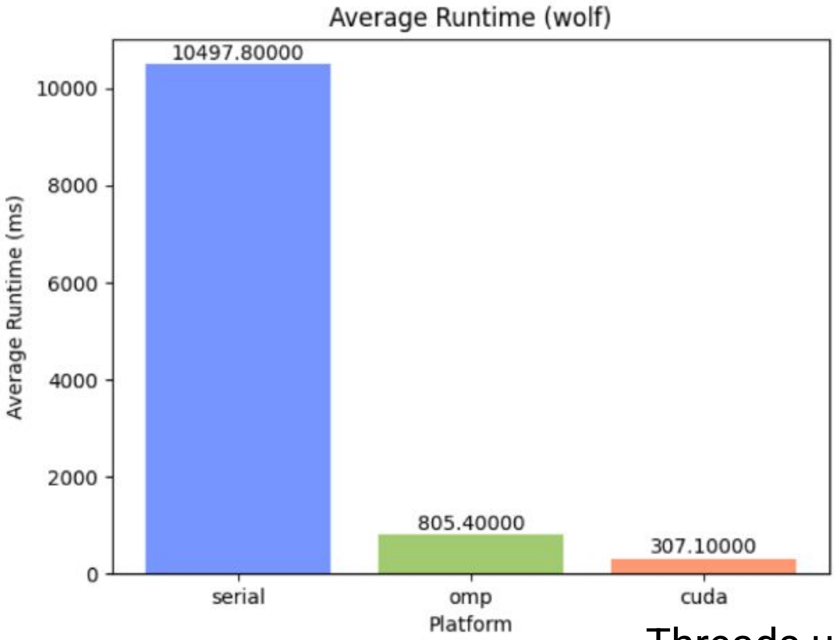
Dimensions : 678 x 446, Size : 12.7 KB



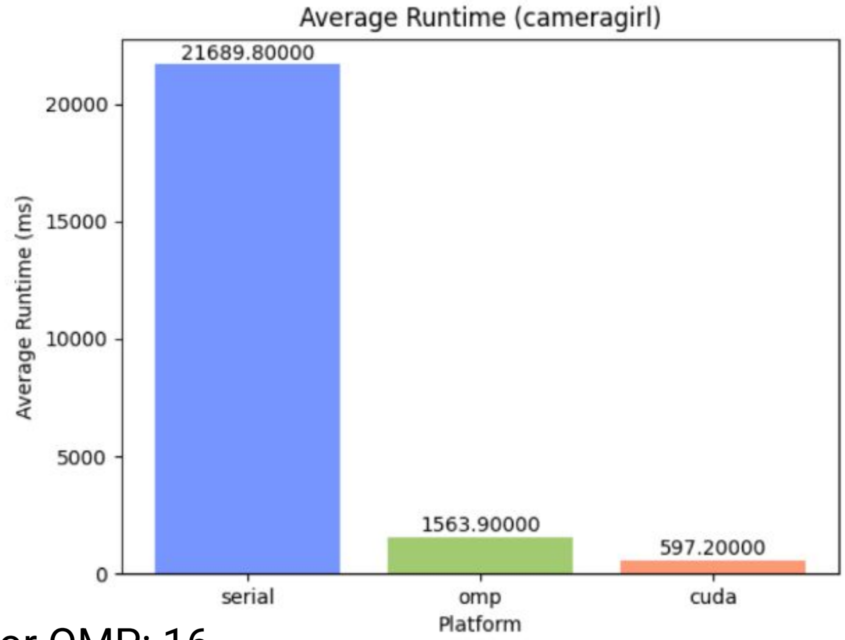
Threads used for OMP: 16
Threads per block used for CUDA: 256

Result - Grayscale (Average Runtimes)

Dimensions : 2048 x 2048, Size : 1 MB



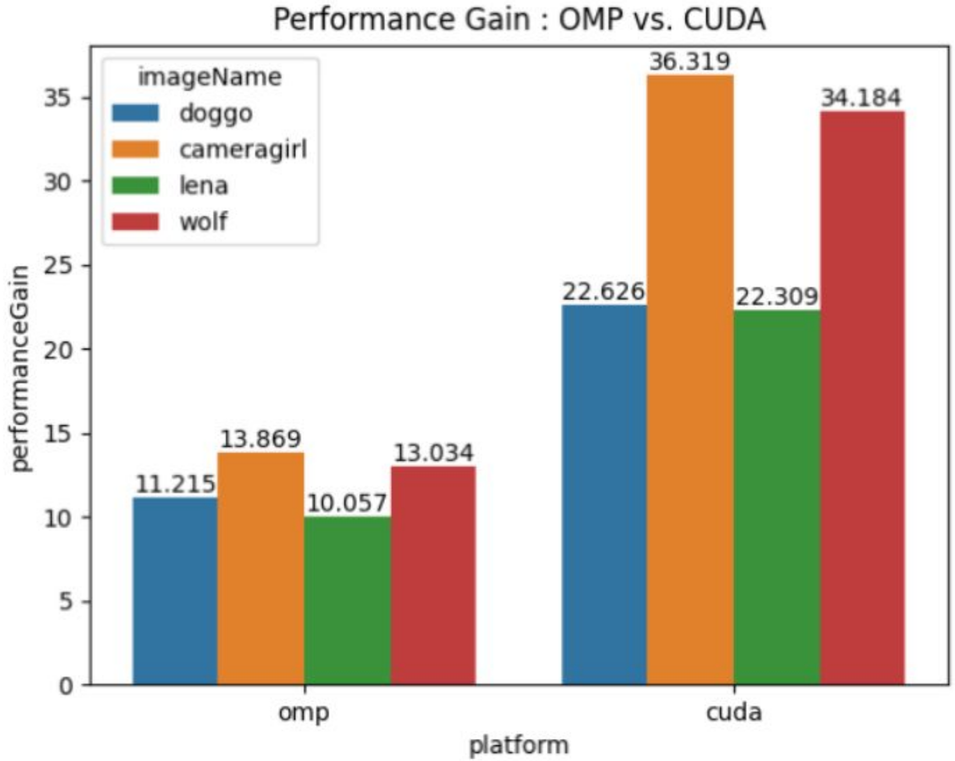
Dimensions : 3000 x 2003, Size : 334 KB



Threads used for OMP: 16

Threads per block used for CUDA: 256

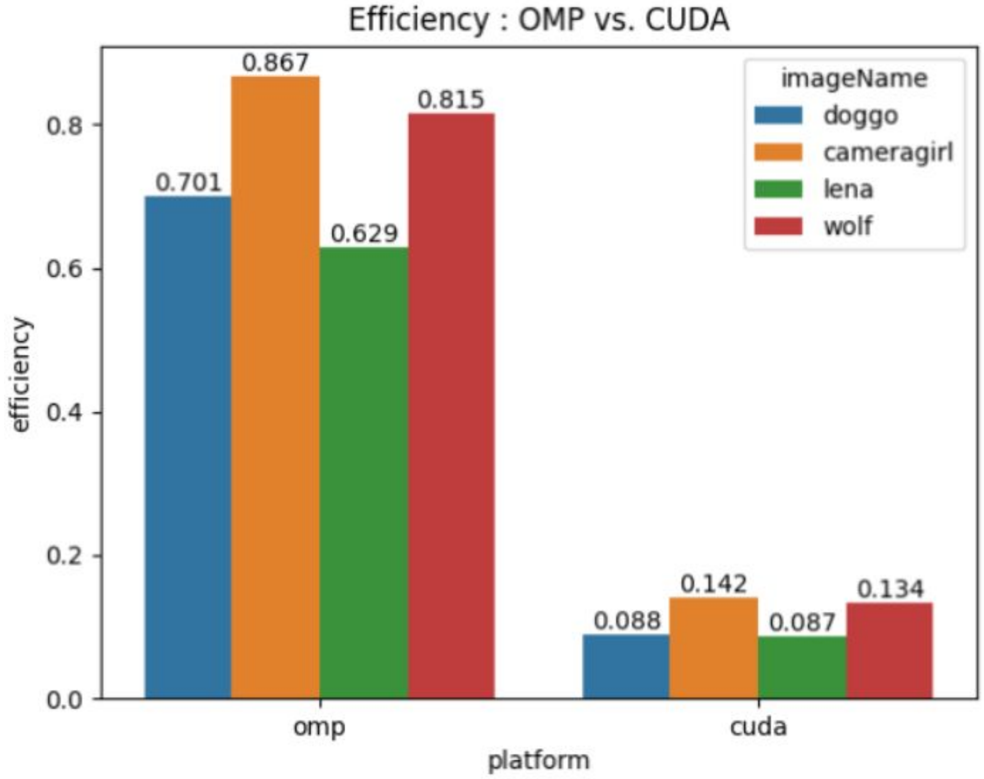
Result - Grayscale (Performance Gain)



$$\text{Performance Gain} = \frac{T_{\text{serial}}}{T_{\text{parallel}}}$$

Threads used for OMP: 16
Threads per block used for CUDA: 256

Result - Grayscale (Efficiency)



$$\text{Efficiency} = \frac{\text{Performance Gain}}{\text{Number of Processors}}$$

Threads used for OMP: 16
Threads per block used for CUDA: 256

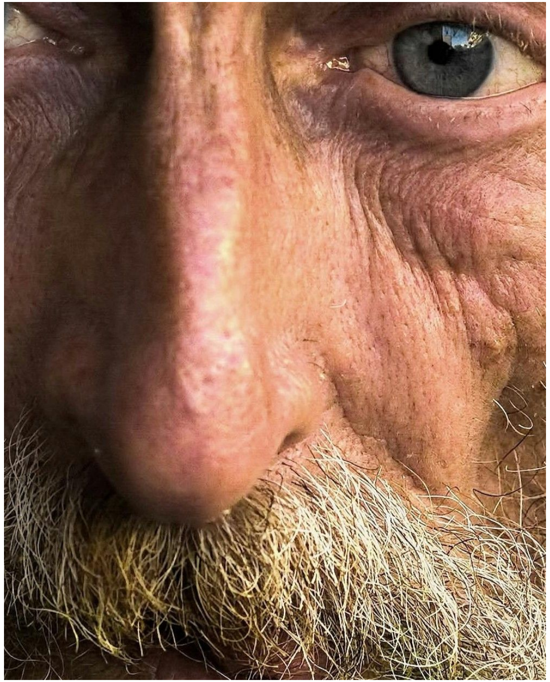
With the success of parallelizing grayscale image enhancement, we proceeded with RGB image enhancement, by applying the same enhancement 3 times, 1 time on each BGR channels, and merging them in the end.

Result - RGB

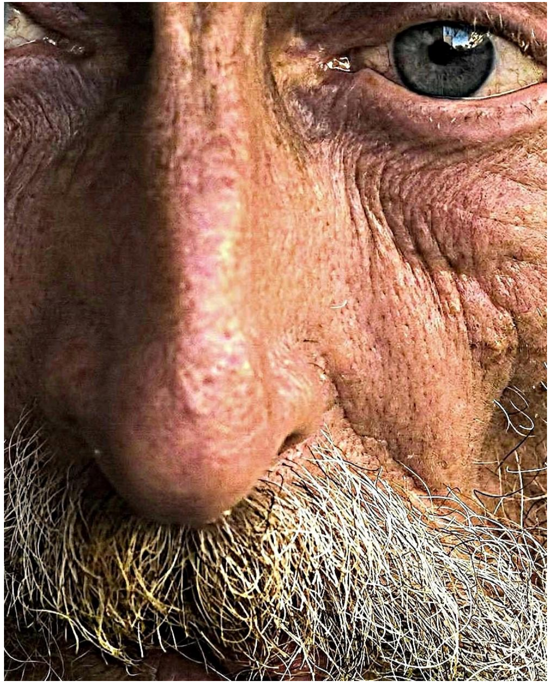

Original
Red Processed
Green Processed
Blue Processed
Final Result

Result - RGB (Before & After)

Before

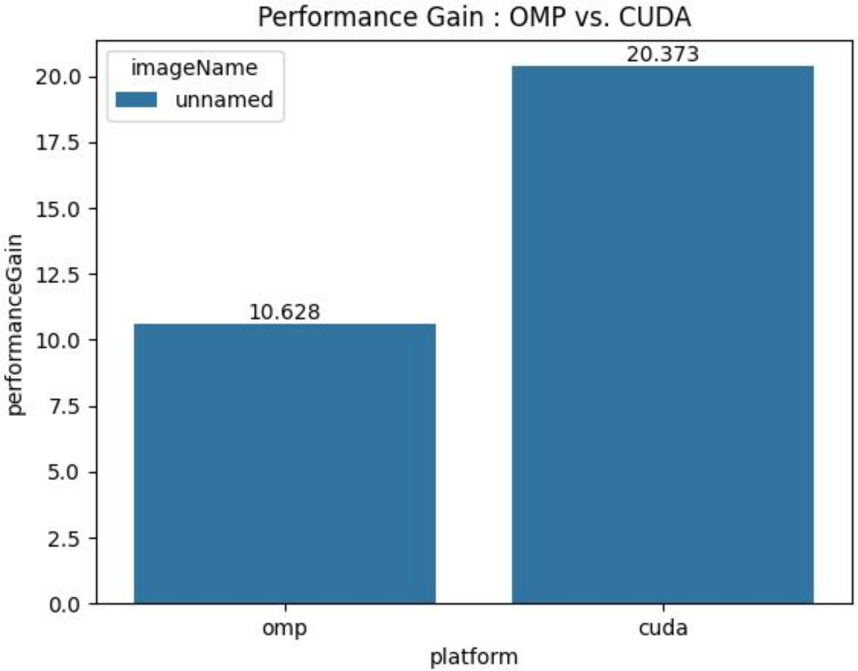


After



Result - RGB (Performance Gain)

Dimensions : 4000 x 5000, Size : 2.74 MB

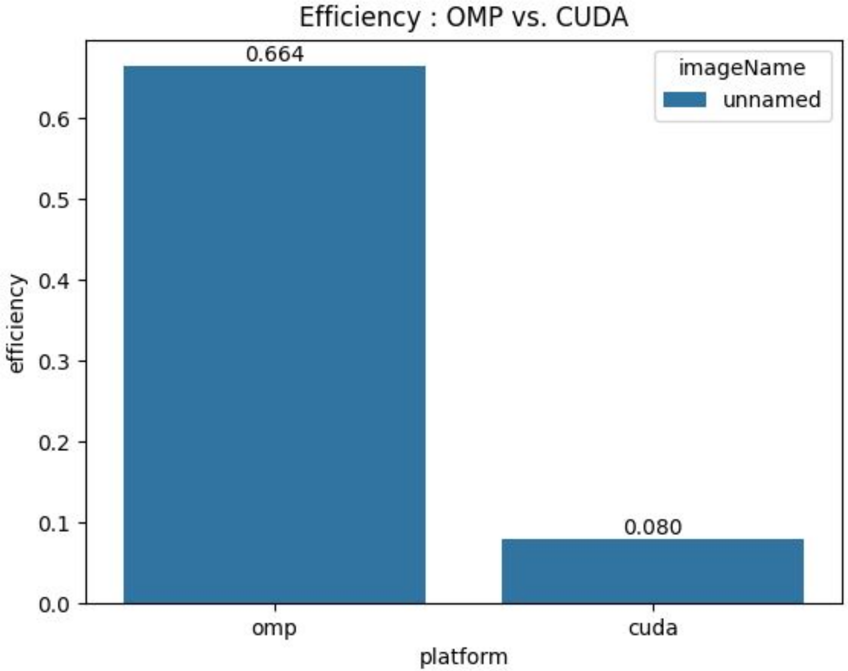


$$\text{Performance Gain} = \frac{T_{\text{serial}}}{T_{\text{parallel}}}$$

Threads used for OMP: 16
Threads per block used for CUDA: 256

Result - RGB (Efficiency)

Dimensions : 4000 x 5000, Size : 2.74 MB



$$\text{Efficiency} = \frac{\text{Performance Gain}}{\text{Number of Processors}}$$

Threads used for OMP: 16

Threads per block used for CUDA: 256