# Lab Assignment 18

## Implement the following Image Processing operations in sequential and parallel using CUDA Programming.

## Gaussian Blur

### Description

Gaussian blur is a widely used image processing operation that helps in reducing image noise and details, thus creating a smoother image. It works by averaging the intensity of pixels in the vicinity of each pixel, weighted by a Gaussian distribution. This weighted averaging process blurs the image while preserving its overall structure.

### Parallelism Insertion

1. Divide the Workload: Split the image processing tasks among multiple threads, each handling a portion of the image.
2. Use GPU-accelerated Operations: Leverage CuPy's GPU-accelerated functions to perform image processing operations on the GPU.
3. Parallel Kernel Launch: Launch a CUDA kernel with multiple threads to execute the processing tasks concurrently on the GPU.
4. Ensure Synchronization: Synchronize the GPU to ensure all threads have completed their tasks before proceeding to the next steps or accessing the processed data.
5. Optimize Memory Usage: Utilize GPU memory efficiently by minimizing data transfers between the CPU and GPU and optimizing memory allocation and deallocation

## Performance Analysis

### Sequential

```
import numpy as np
from scipy.signal import convolve2d
from PIL import Image
import os
import time

def process_image(image_array):
    def gaussian_kernel(size, sigma=1):
        kernel_1D = np.linspace(-(size // 2), size // 2, size)
        for i in range(size):
            kernel_1D[i] = np.exp(-0.5 * (kernel_1D[i] / sigma) ** 2)
        kernel_2D = np.outer(kernel_1D, kernel_1D)
        kernel_2D /= kernel_2D.sum()
        return kernel_2D

    kernel_size = 5
    gaussian_kernel_array = gaussian_kernel(kernel_size)
    blurred_image = convolve2d(image_array, gaussian_kernel_array, mode='same', boundary='wrap')

    return blurred_image

directory = "/content/drive/MyDrive/NNDL/dog cat/dog cat/train/cats"
num_images = 0
start_time = time.time()

image_paths = [os.path.join(directory, filename) for filename in os.listdir(directory) if filename.endswith(".jpg")]
num_images = len(image_paths)

for image_path in image_paths:
    image_array = np.array(Image.open(image_path).convert("L"))
    image_blurred = process_image(image_array)

total_time_sequential = time.time() - start_time

print("Number of images processed in sequence:", num_images)
print("Time taken for sequential processing:", total_time_sequential, "seconds")
```

```
Number of images processed in sequence: 279
Time taken for sequential processing: 38.25399899482727 seconds
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

### Parallel

```
import cupy as cp
from PIL import Image
import os
import time

def process_image(image_array):
    def gaussian_kernel(size, sigma=1):
        kernel_1D = cp.linspace(-(size // 2), size // 2, size)
        for i in range(size):
            kernel_1D[i] = cp.exp(-0.5 * (kernel_1D[i] / sigma) ** 2)
        kernel_2D = cp.outer(kernel_1D, kernel_1D)
        kernel_2D /= kernel_2D.sum()
        return kernel_2D

    kernel_size = 5
    gaussian_kernel_array = gaussian_kernel(kernel_size)
    blurred_image = cp.asarray(Image.fromarray(cp.asnumpy(image_array)).convert("L"))

    return blurred_image

num_images = 0
start_time = time.time()

image_paths = [os.path.join(directory, filename) for filename in os.listdir(directory) if filename.endswith(".jpg")]
image_arrays = [cp.array(Image.open(image_path).convert("L")) for image_path in image_paths]
processed_images = [process_image(image_array) for image_array in image_arrays]

cp.cuda.Device().synchronize()

total_time_parallel = time.time() - start_time
num_images = len(image_paths)

print("Number of images processed in parallel:", num_images)
print("Time taken for parallel processing:", total_time_parallel, "seconds")
```

```
Number of images processed in parallel: 279
Time taken for parallel processing: 2.9992127418518066 seconds
```

## FFT - Fast Fourier Transform

### Description

The Fast Fourier Transform (FFT) is a widely used algorithm for efficiently computing the Discrete. It transforms a signal from its time or spatial domain into its frequency domain, revealing the frequency components present in the signal. FFT has numerous applications in signal processing, image processing, data compression, and more

### Parallelism Insertion

1. Divide and Conquer: Divide the input data into smaller chunks and distribute them among multiple threads on the GPU.
2. Utilize GPU-accelerated Libraries: Leverage GPU-accelerated libraries like CuPy, which provide efficient implementations of FFT algorithms optimized for GPU execution.
3. Parallel Kernel Launch: Launch a CUDA kernel with multiple threads to perform parallel FFT computation on the GPU. Each thread processes a portion of the input data independently.
4. Ensure Synchronization: Synchronize the GPU to ensure all threads have completed their FFT computations before proceeding to the next steps or accessing the results.
5. Optimize Memory Usage: Optimize memory access patterns and data transfers between the CPU and GPU to minimize overhead and maximize throughput.

## Performance Analysis

### Sequential

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import fftpack
from PIL import Image
import os
import time

num_images = 0
start_time = time.time()
directory = "/content/drive/MyDrive/NNDL/dog cat/dog cat/train/cats"

for filename in os.listdir(directory):
    if filename.endswith(".jpg"):
        num_images += 1

        image = Image.open(os.path.join(directory, filename)).convert("L")
        image_array = np.array(image)

        fft_image = fftpack.fft2(image_array)
        fft_image_shifted = fftpack.fftshift(fft_image)

        rows, cols = image_array.shape
        center_row, center_col = rows // 2, cols // 2
        radius = 20
        high_pass_filter = np.ones((rows, cols))
        mask = np.zeros((rows, cols))
        mask[center_row - radius:center_row + radius, center_col - radius:center_col + radius] = 1
        high_pass_filter -= mask

        filtered_image_fft = fft_image_shifted * high_pass_filter

        filtered_image = np.abs(fftpack.ifft2(fftpack.ifftshift(filtered_image_fft)))

total_time = time.time() - start_time

print("Number of images processed in sequence:", num_images)
print("Time taken for sequential processing:", total_time, "seconds")
```

```
Number of images processed in sequence: 279
Time taken for sequential processing: 28.283076286315918 seconds
```

## Parallel

```
import os
import time
import cupy as cp
from PIL import Image


num_images = 0
start_time = time.time()

def process_image(image_array):
    global num_images
    num_images += 1

    fft_image = cp.fft.fft2(image_array)
    fft_image_shifted = cp.fft.fftshift(fft_image)

    rows, cols = image_array.shape
    center_row, center_col = rows // 2, cols // 2
    radius = 20
    high_pass_filter = cp.ones((rows, cols))
    mask = cp.zeros((rows, cols))
    mask[center_row - radius:center_row + radius, center_col - radius:center_col + radius] = 1
    high_pass_filter -= mask

    filtered_image_fft = fft_image_shifted * high_pass_filter

    filtered_image = cp.abs(cp.fft.ifft2(cp.fft.ifftshift(filtered_image_fft)))
    return filtered_image

image_paths = [os.path.join(directory, filename) for filename in os.listdir(directory) if filename.endswith(".jpg")]
image_arrays = [cp.array(Image.open(image_path).convert("L")) for image_path in image_paths]
processed_images = [process_image(image_array) for image_array in image_arrays]

cp.cuda.Device().synchronize()

total_time_parallel = time.time() - start_time

print("Number of images processed in parallel:", num_images)
print("Time taken for parallel processing:", total_time_parallel, "seconds")
```

```
Number of images processed in parallel: 279
Time taken for parallel processing: 4.5335447788238525 seconds
```