TASK 4:

The C code uses CUDA to accelerate a 3x3 box blur filter on a PNG image. It decodes the input image using the lodepng library, allocates GPU memory, and launches a CUDA kernel for parallelized image processing. After synchronizing threads, the blurred image is transferred back to the host and saved as a new PNG file. The code efficiently handles GPU memory and produces an output image with the applied blur effect.

1. Reading in an image file into a single or 2D array

The lodepng library to decode a PNG image file named "input.png". The image data is stored in the image array, and the width and height of the image are obtained.

```
    unsigned char *image; // Pointer to image data
    unsigned int width, height; // Width and height of the image

    // Decode PNG
    unsigned int error = lodepng_decode32_file(&image, &width, &height,
"input.png");
    if (error)
    {
        fprintf(stderr, "Error %u: %s\n", error, lodepng_error_text(error));
        return 1;
    }
```

2. Allocating the correct amount of memory on the GPU based on input data. Memory is freed once used:

Device memory (d_input and d_output) is allocated using cudaMalloc to store the image data on the GPU. The memory size is determined based on the image dimensions and the number of color channels (4 for RGBA).

```
    // Allocate device memory for the image
    unsigned char *d_input, *d_output;
    cudaMalloc((void **)&d_input, sizeof(unsigned char) * width * height * 4);
    cudaMalloc((void **)&d_output, sizeof(unsigned char) * width * height * 4);
```

The cudaFree function is used later to release this memory.

```
    // Free device memory
    cudaFree(d_input);
    cudaFree(d_output);
```

3. Applying Box filter on image in the kernel function:

The CUDA kernel function boxBlur operates on the GPU and applies a 3x3 box blur filter to each pixel in the image. It calculates the average color values of the 3x3 neighborhood for each pixel.

```
// CUDA kernel for applying a box blur filter to an image
__global__ void boxBlur(unsigned char *input, unsigned char *output, unsigned int
width, unsigned int height)
{
    // Calculate the pixel coordinates in the output image based on block and
thread indices
    int p_i = blockIdx.y * blockDim.y + threadIdx.y;
    int p_j = blockIdx.x * blockDim.x + threadIdx.x;

    // Check if the calculated coordinates are within the image dimensions
    if (p_i < height && p_j < width)
    {
        // Initialize variables to accumulate color channels and count
neighboring pixels
        int r = 0, g = 0, b = 0;
        int count = 0;

        // Iterate over the 3x3 neighborhood centered at the current pixel
        for (int i = -1; i <= 1; ++i)
        {
            for (int j = -1; j <= 1; ++j)
            {
                // Calculate the coordinates of the neighboring pixel
                int neighbor_i = p_i + i;
                int neighbor_j = p_j + j;

                // Check if the neighboring pixel is within the image boundaries
                if (neighbor_i >= 0 && neighbor_i < height && neighbor_j >= 0 &&
neighbor_j < width)
                {
                    // Calculate the index of the neighboring pixel in the input
image
                    int index = 4 * (neighbor_i * width + neighbor_j);
                    // Accumulate color channels and increment the count
                    r += input[index];
                    g += input[index + 1];
                    b += input[index + 2];
                    count++;
                }
            }
        }

        // Calculate the index of the current pixel in the output image
        int index = 4 * (p_i * width + p_j);
        // Compute the average color values for the 3x3 neighborhood and update
the output image
        output[index] = r / count;
        output[index + 1] = g / count;
        output[index + 2] = b / count;
        // Copy the alpha channel from the input to the output (unchanged)
        output[index + 3] = input[index + 3];
    }
}
```

4. Return blurred image data from the GPU to the CPU:

After the GPU processing is complete, the blurred image data (d_output) is copied back from the GPU to the host (image) using cudaMemcpyDeviceToHost.

```
// Copy the result back from device to host
    cudaMemcpy(image, d_output, sizeof(unsigned char) * width * height * 4,
cudaMemcpyDeviceToHost);
```

5. Outputting the correct image with Box Blur applied as a file:

The blurred image data stored in the image array is encoded and saved as a new PNG file named "output.png" using the lodepng_encode32_file function. This file represents the original image with the box blur applied.

```
    error = lodepng_encode32_file("output.png", image, width, height);
    if (error)
    {
        fprintf(stderr, "Error %u: %s\n", error, lodepng_error_text(error));
        return 1;
    }
```