

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное учреждение высшего образования  
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»



Инженерная школа информационных технологий и робототехники  
Направление подготовки 09.04.01 Информатика и вычислительная техника  
Отделение Информационных технологий

**Отчет по Практической работе №2**  
**по дисциплине «Параллельные и высокопроизводительные вычисления»**

Тема работы
<b>Реализация многопоточных вычислений на GPU</b>

**Вариант 5**

Студент

Группа	ФИО	Подпись	Дата
8BM22	Ямкин Н.Н.		

Руководитель

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ	Аксёнов С.В.	к.т.н., доцент		

## Ход работы

### 1 Программа А

#### 1.1 Задание

Произвести тестирование программы для разных размеров обрабатываемых файлов изображений. Для тестирования взять файлы размерами: 10240 x 7680, 12800 x 9600, 20480 x 15360. Получить среднее значение работы процедуры обработки каждого изображения при троекратном перезапуске программы.

Загрузить цветное изображение.

Получить значения интенсивности  $I_v = (Red_v + Green_v + Blue_v)/3$ , где  $I_v$  – интенсивность пикселя  $v$ ,  $Red_v$  – значение красной компоненты пикселя  $v$ ,  $Green_v$  – значение зелёной компоненты пикселя  $v$ ,  $Blue_v$  – значение синей компоненты пикселя  $v$ .

Установить значение скалярной величины - порога Threshold (любое число от 1 до 255). Те значения интенсивности, которые меньше порогового значения установить в 0, а которые больше Threshold установить в 1.

Выполнить операцию наращивания (диляции/ дилатации) [https://habr.com/ru/post/113626/ или https://intuit.ru/studies/courses/10621/1105/lecture/17989?page=4] над полученной матрицей из 0 и 1. Примечание: нужно задать шаг наращивания (любое значение от 1, 2 или 3). Получить изображение из результата путем установки вместо значений 0 – пикселей черного цвета (0, 0, 0), и вместо значений 1 – пикселей белого цвета. Сохранить результат в файл.

## 1.2 Листинг программы с комментариями

```
from google.colab import drive
drive.mount('/content/gdrive')
%%bash
pip install pycuda

import numpy as np
import time
import pycuda.autotinit
from pycuda import gpuarray
import cv2
import matplotlib.pyplot as plt

image = cv2.imread('/content/gdrive/MyDrive/tropa_les_derevia_944568_1024x768.jpg')
plt.imshow(image)

image_2 = cv2.imread('/content/gdrive/MyDrive/tropa_les_derevia_10240x7680.jpg')
image_2.shape

image_3 = cv2.imread('/content/gdrive/MyDrive/цветок_лепестки_белый_12800_9600.jpg')
image_3.shape

print(f'File Sizes = {image.shape}')
image_height = image.shape[0]
print(f'Image Height = {image_height}')
image_width = image.shape[1]
print(f'Image Width = {image_width}')
num_of_channels = image.shape[2]
print(f'Number of Channels = {num_of_channels}')
num_of_pixels = image_height * image_width
print(f'Number of Pixels = {num_of_pixels}')
image_size = image_height * image_width * num_of_channels
```

```

print(f'Image Size = {image_size} uint8')

input_image = image.astype(np.int32)
input_image = input_image.reshape(image_size)
print(type(input_image[0]))

# Компиляция ядра
from pycuda.compiler import SourceModule
grey_scale_plus_binarization_kernel = SourceModule("""
__global__ void grey_scale_plus_binarization_kernel(int *input, int *output, int image_height, int image_width, int
threshold)
{
    int x = blockDim.x * blockIdx.x + threadIdx.x;
    int y = blockDim.y * blockIdx.y + threadIdx.y;

    if (x<image_width)
        if (y<image_height)
        {
            int i = y * image_width + x;
            int r = input[3*i];
            int g = input[3*i + 1];
            int b = input[3*i + 2];
            int gray_pixel = (r + g + b)/3;

            if (gray_pixel > threshold)
            {
                output[3*i] = 255;
                output[3*i+1] = 255;
                output[3*i+2] = 255;
            }
            else

```

```

        {
            output[3*i] = 0;
            output[3*i+1] = 0;
            output[3*i+2] = 0;
        }
    }
}"""

grey_scale_plus_binarization_kernel =
grey_scale_plus_binarization_kernel.get_function('grey_scale_plus_binarization_kernel')

block_2D_size_x = 32
block_2D_size_y = 32
block_2D_size_z = 1

grid_2D_size_x = round(image_width/block_2D_size_x) + 1
grid_2D_size_y = round(image_height/block_2D_size_y) + 1
grid_2D_size_z = 1
print(f'Block size = ({block_2D_size_x}, {block_2D_size_y}, {block_2D_size_z}) Threads')
print(f'Grid size = ({grid_2D_size_x}, {grid_2D_size_y}, {grid_2D_size_z}) Blocks')

# Выделение памяти на GPU
device_input_image = gpuarray.to_gpu(input_image)
device_output_image = gpuarray.empty_like(device_input_image)

# Выполнение ядра
grey_scale_plus_binarization_kernel(device_input_image, device_output_image,
                                     np.int32(image_height), np.int32(image_width), np.int32(42),
                                     block = (block_2D_size_x, block_2D_size_y, block_2D_size_z),
                                     grid = (grid_2D_size_x, grid_2D_size_y, grid_2D_size_z))

```

```

# Получение ссылки на функцию ядра
host_output_image = device_output_image.get()

host_output_image = host_output_image.astype(np.uint8)
host_output_image = host_output_image.reshape(image.shape)

plt.imshow(host_output_image[:, :, 0], cmap = 'gray')

# Дилатация делается на одноканальном изображении
binarized_image = host_output_image[:, :, 0]

# Компиляция ядра
dilate_kernel = SourceModule("""
__global__ void dilate_kernel(unsigned char *input, unsigned char *output, int image_height, int image_width, int
kernel_size)
{
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;

    if (x >= image_width || y >= image_height)
    {
        return; // гарантируем, чтобы потоки находятся внутри границ изображения
    }

    int radius = kernel_size / 2;
    unsigned char max_value = 0;

    for (int i = -radius; i <= radius; i++)
    {
        for (int j = -radius; j <= radius; j++)
        {

```

```

        int pixel_x = min(max(x + i, 0), image_width - 1);
        int pixel_y = min(max(y + j, 0), image_height - 1);

        int index = pixel_y * image_width + pixel_x;
        unsigned char pixel = input[index];

        if (pixel > max_value)
        {
            max_value = pixel;
        }
    }

    int output_index = y * image_width + x;
    output[output_index] = max_value;
}
""")
dilate_kernel = dilate_kernel.get_function('dilate_kernel')

# Выделение памяти на GPU
device_input_image = gpuarray.to_gpu(binarized_image)
device_output_image = gpuarray.empty_like(device_input_image)

# Выполнение ядра
dilate_kernel(device_input_image, device_output_image, np.int32(image_height), np.int32(image_width), np.int32(3),
              block = (block_2D_size_x, block_2D_size_y, block_2D_size_z),
              grid = (grid_2D_size_x, grid_2D_size_y, grid_2D_size_z))

# Получение ссылки на функцию ядра
dilated_image = device_output_image.get()

```

```

plt.imshow(dilated_image, cmap = 'gray')

#### Тестирование

elapsed_time = []
for _ in range(3):

    image_height = image.shape[0]
    image_width = image.shape[1]
    num_of_channels = image.shape[2]
    image_size = image_height * image_width * num_of_channels

    input_image = image.astype(np.int32)
    input_image = input_image.reshape(image_size)

    device_input_image = gpuarray.to_gpu(input_image)
    device_output_image = gpuarray.empty_like(device_input_image)

    block_2D_size_x = 32
    block_2D_size_y = 32
    block_2D_size_z = 1

    grid_2D_size_x = round(image_width/block_2D_size_x) + 1
    grid_2D_size_y = round(image_height/block_2D_size_y) + 1
    grid_2D_size_z = 1

    start_time = time.time()

    grey_scale_plus_binarization_kernel(device_input_image, device_output_image,
                                         np.int32(image_height), np.int32(image_width), np.int32(42),
                                         block = (block_2D_size_x, block_2D_size_y, block_2D_size_z),

```



```

        grid = (grid_2D_size_x, grid_2D_size_y, grid_2D_size_z))

host_output_image = device_output_image.get()
host_output_image = host_output_image.astype(np.uint8)
host_output_image = host_output_image.reshape(image.shape)

binarized_image = host_output_image[:, :, 0]

device_input_image = gpuarray.to_gpu(binarized_image)
device_output_image = gpuarray.empty_like(device_input_image)

dilate_kernel(device_input_image, device_output_image, np.int32(image_height), np.int32(image_width),
np.int32(3),
        block = (block_2D_size_x, block_2D_size_y, block_2D_size_z),
        grid = (grid_2D_size_x, grid_2D_size_y, grid_2D_size_z))

dilated_image = device_output_image.get()

elapsed_time.append(time.time() - start_time)

print(f'Среднее время обработки изображения {np.mean(elapsed_time)} секунд')

```

### 1.3 Описание аппаратной базы

Данная программа выполнялась в среде Google Colab, и для получения характеристик аппаратной базы использовался следующий блок кода:

```
import pycuda.driver as drv
drv.init()
print(f'Number of GPU Devices Detected: {drv.Device.count()}')

gpu_device = drv.Device(0)
print(f'GPU Device name: {gpu_device.name()}')
print(f'GPU Device Compute Capability: {gpu_device.compute_capability()}')
print(f'GPU Global Memory Size = {gpu_device.total_memory()//1024**2}
Mbytes')

device_attributes_keys = gpu_device.get_attributes().keys()
device_attributes_values = gpu_device.get_attributes().values()
count = len(list(device_attributes_keys))
device_attributes_keys = list(device_attributes_keys)
device_attributes_values = list(device_attributes_values)

device_attributes = {}
for i in range(count):
    device_attributes[str(device_attributes_keys[i])] =
device_attributes_values[i]

print('GPU Device Technical Specification:')
for i in device_attributes.keys():
    print(f'{i} : {device_attributes[i]}')
```

Получили следующее:

```
GPU Device name: Tesla T4
GPU Device Compute Capability: (7, 5)
GPU Global Memory Size = 15101 Mbytes

GPU Device Technical Specification:
ASYNC_ENGINE_COUNT : 3
CAN_MAP_HOST_MEMORY : 1
CAN_USE_HOST_POINTER_FOR_REGISTERED_MEM : 1
CLOCK_RATE : 1590000
COMPUTE_CAPABILITY_MAJOR : 7
COMPUTE_CAPABILITY_MINOR : 5
COMPUTE_MODE : DEFAULT
COMPUTE_PREEMPTION_SUPPORTED : 1
CONCURRENT_KERNELS : 1
CONCURRENT_MANAGED_ACCESS : 1
DIRECT_MANAGED_MEM_ACCESS_FROM_HOST : 0
ECC_ENABLED : 1
GENERIC_COMPRESSION_SUPPORTED : 0
GLOBAL_L1_CACHE_SUPPORTED : 1
GLOBAL_MEMORY_BUS_WIDTH : 256
GPU_OVERLAP : 1
HANDLE_TYPE_POSIX_FILE_DESCRIPTOR_SUPPORTED : 1
HANDLE_TYPE_WIN32_HANDLE_SUPPORTED : 0
HANDLE_TYPE_WIN32_KMT_HANDLE_SUPPORTED : 0
HOST_NATIVE_ATOMIC_SUPPORTED : 0
INTEGRATED : 0
KERNEL_EXEC_TIMEOUT : 0
L2_CACHE_SIZE : 4194304
LOCAL_L1_CACHE_SUPPORTED : 1
MANAGED_MEMORY : 1
MAX_BLOCKS_PER_MULTIPROCESSOR : 16
MAX_BLOCK_DIM_X : 1024
MAX_BLOCK_DIM_Y : 1024
MAX_BLOCK_DIM_Z : 64
MAX_GRID_DIM_X : 2147483647
MAX_GRID_DIM_Y : 65535
MAX_GRID_DIM_Z : 65535
MAX_PERSISTING_L2_CACHE_SIZE : 0
MAX_PITCH : 2147483647
MAX_REGISTERS_PER_BLOCK : 65536
MAX_REGISTERS_PER_MULTIPROCESSOR : 65536
MAX_SHARED_MEMORY_PER_BLOCK : 49152
MAX_SHARED_MEMORY_PER_BLOCK_OPTIN : 65536
MAX_SHARED_MEMORY_PER_MULTIPROCESSOR : 65536
MAX_THREADS_PER_BLOCK : 1024
MAX_THREADS_PER_MULTIPROCESSOR : 1024
MEMORY_CLOCK_RATE : 5001000
MEMORY_POOLS_SUPPORTED : 1
MULTIPROCESSOR_COUNT : 40
MULTI_GPU_BOARD : 0
MULTI_GPU_BOARD_GROUP_ID : 0
PAGEABLE_MEMORY_ACCESS : 0
PAGEABLE_MEMORY_ACCESS_USES_HOST_PAGE_TABLES : 0
PCI_BUS_ID : 0
PCI_DEVICE_ID : 4
PCI_DOMAIN_ID : 0
READ_ONLY_HOST_REGISTER_SUPPORTED : 1
RESERVED_SHARED_MEMORY_PER_BLOCK : 0
SINGLE_TO_DOUBLE_PRECISION_PERF_RATIO : 32
STREAM_PRIORITIES_SUPPORTED : 1
SURFACE_ALIGNMENT : 512
TCC_DRIVER : 0
TEXTURE_ALIGNMENT : 512
TEXTURE_PITCH_ALIGNMENT : 32
TOTAL_CONSTANT_MEMORY : 65536
UNIFIED_ADDRESSING : 1
WARP_SIZE : 32
```

Рисунок 1 – Характеристики GPU в Google Colab

Видеокарта NVIDIA Tesla T4 имеет объем памяти равный 16 ГБ и базовую частоту графического процессора 1005 МГц.

## 1.4 Примеры входного и выходного изображения

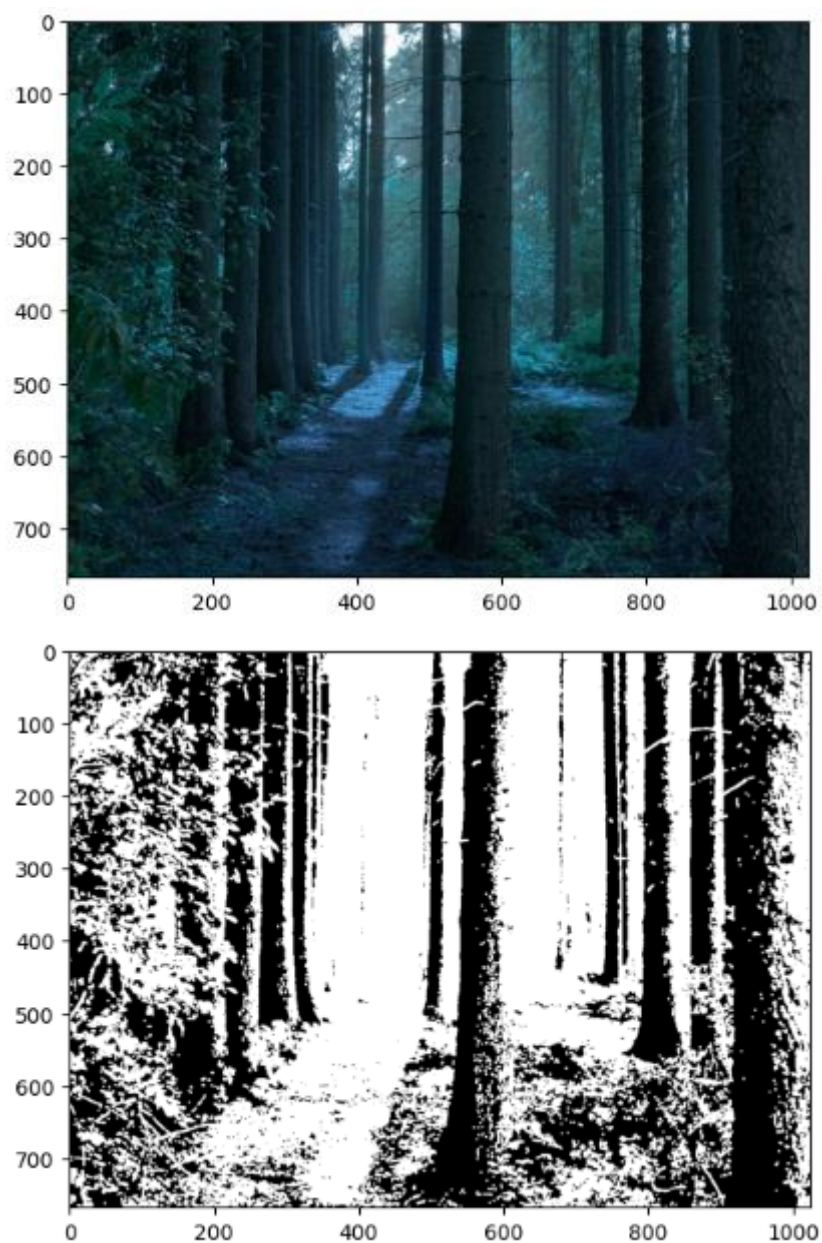


Рисунок 2 – Изображение 768×1024

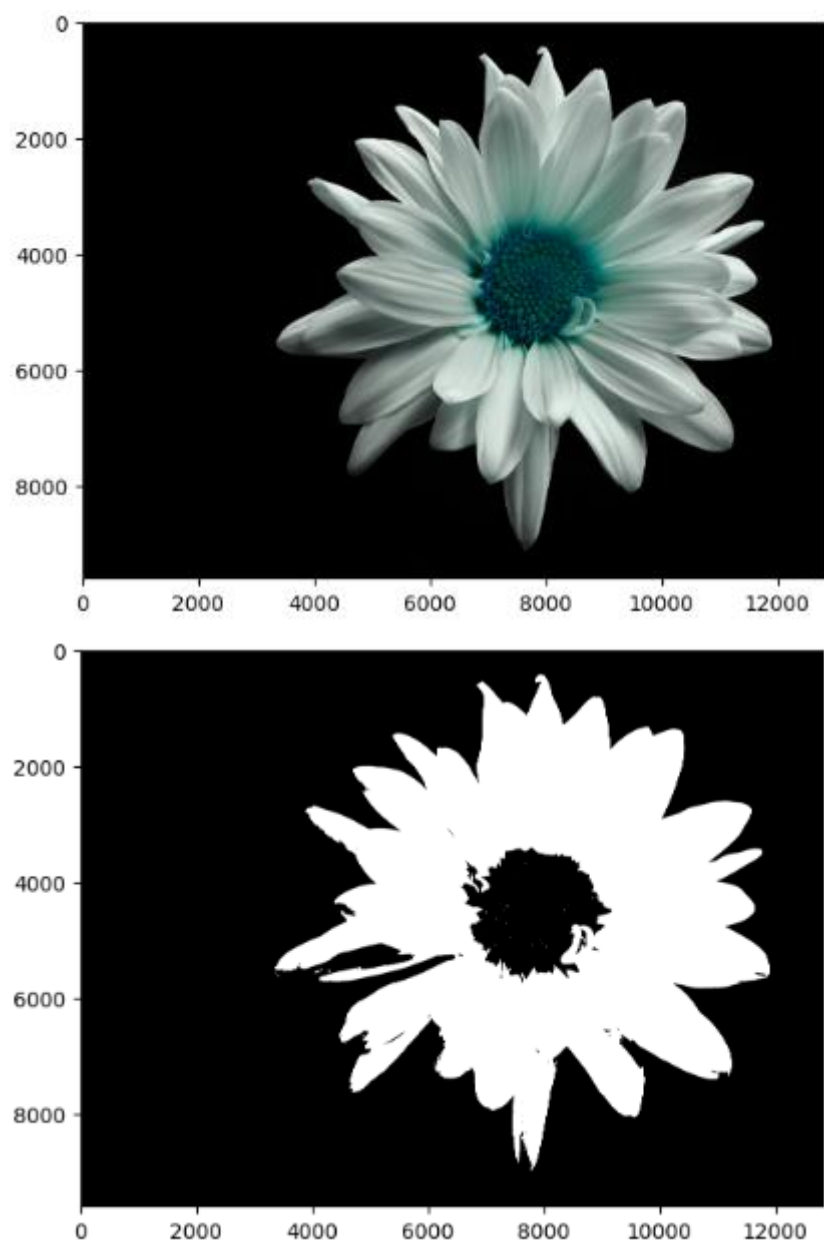


Рисунок 3 – Изображение 9600×12800

### 1.5 Производительность программы

Таблица 1 – Среднее время обработки изображений

	768×1024	7680×10240	9600×12800
Среднее время, с	0.085914	8.016286	12.688158

## 2 Программа В

### 2.1 Задание

Произвести тестирование программы для разных размеров обрабатываемых файлов изображений. Для тестирования взять файлы размерами: 10240 x 7680, 12800 x 9600, 20480 x 15360. Получить среднее значение работы процедуры обработки каждого изображения при троекратном перезапуске программы.

Получить значения интенсивности  $I_v = (Red_v + Green_v + Blue_v)/3$ , где  $I_v$  – интенсивность пикселя  $v$ ,  $Red_v$  – значение красной компоненты пикселя  $v$ ,  $Green_v$  – значение зелёной компоненты пикселя  $v$ ,  $Blue_v$  – значение синей компоненты пикселя  $v$ . Найти максимальное значение интенсивности в массиве, и все цветовые компоненты и значение интенсивности в пикселе  $v$  установить в нуль, если значение  $I_v < 0.1 \times I_{max}$

Цветовые каналы для пикселей с ненулевой модифицированной интенсивностью изменяются, и дополнительно выделяется жёлтый канал, согласно следующим выражениям для красного, зелёного, синего и жёлтого цветовых компонент соответственно:

$$R_v = Red_v - (Green_v + Blue_v)/2,$$

$$G_v = Green_v - (Red_v + Blue_v)/2,$$

$$B_v = Blue_v - (Green_v + Red_v)/2,$$

$$Y_v = Red_v + Green_v - 2 \cdot (|Red_v - Green_v| + Blue_v)$$

Получить матрицы модифицированной интенсивности, а также модифицированные цветовые каналы для красного, зеленого, синего и желтого канала. Сохранить полученные матрицы, как полутоновые изображения.

## 2.2 Листинг программы с комментариями

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

%%bash
pip install pyopencl

!sudo apt update
!sudo apt purge *nvidia* -y
!sudo apt install nvidia-driver-530 -y

import pyopencl as cl
import numpy as np
import pyopencl as cl
from skimage import io
import time

# Создание контекста OpenCL и выбор платформы и устройства
platform = cl.get_platforms()[0]
device = platform.get_devices()[0]
context = cl.Context([device])
queue = cl.CommandQueue(context)

#### Тестовое изображение с разрешением 768x1024

image = cv2.imread('/content/gdrive/MyDrive/tropa_les_derevia_944568_1024x768.jpg')
plt.imshow(image)

#### Изображение с разрешением 10240x7680

image = cv2.imread('/content/gdrive/MyDrive/tropa_les_derevia_10240x7680.jpg')
image.shape
```

```

#### Изображение с разрешением 12800x9600

image = cv2.imread('/content/gdrive/MyDrive/цветок_лепестки_белый_12800_9600.jpg')
image.shape

print(10*','=', 'INPUT IMAGE', 10*'=')
print(f'File Sizes = {image.shape}')
image_height = image.shape[0]
print(f'Image Height = {image_height}')
image_width = image.shape[1]
print(f'Image Width = {image_width}')
num_of_channels = image.shape[2]
print(f'Number of Channels = {num_of_channels}')
num_of_pixels = image_height * image_width
print(f'Number of Pixels = {num_of_pixels}')
image_size = image_height * image_width * num_of_channels
print(f'Image Size = {image_size} uint8')

print(10*','=', 'OUTPUT IMAGE', 10*'=')
print(f'File Sizes = {image.shape}')
image_height = image.shape[0]
print(f'Image Height = {image_height}')
image_width = image.shape[1]
print(f'Image Width = {image_width}')
output_num_of_channels = 4
print(f'Number of Channels = {output_num_of_channels}')
num_of_pixels = image_height * image_width
print(f'Number of Pixels = {num_of_pixels}')
output_image_size = image_height * image_width * output_num_of_channels
print(f'Image Size = {output_image_size} uint8')

```



```

input_image = image.astype(np.int32)
input_image = input_image.reshape(image_size)
print(type(input_image[0]))

output_image = np.zeros((image_height,image_width,output_num_of_channels),dtype = np.int32)
print(output_image.shape)
output_image = output_image.reshape(output_image_size)
print(type(output_image[0]))
print(output_image.shape)

# Определение исходного кода для ядра
grey_scale_program = cl.Program(context, """
__kernel void grey_scale(__global int *input, __global int *output, __global int *max_intensity, int num_of_pixels)
{

// Получаем глобальный идентификатор
int i = get_global_id(0);
if (i<num_of_pixels)
{

    // расчет интенсивности пикселя
    int r = input[3*i];
    int g = input[3*i + 1];
    int b = input[3*i + 2];
    int gray_pixel = (r + g + b)/3;

    // поиск максимального значения интенсивности
    atomic_max(max_intensity, gray_pixel);

    // синхронизация глобальной памяти

```

```

mem_fence(CLK_GLOBAL_MEM_FENCE);

// получаем модифицированное 4 канальное изображение
if (gray_pixel < *max_intensity * 0.1)
{
    gray_pixel = 0;
    output[4*i] = 0;
    output[4*i + 1] = 0;
    output[4*i + 2] = 0;
    output[4*i + 3] = 0;
}
else
{
    output[4*i] = r - (g + b)/2;
    output[4*i + 1] = g - (r + b)/2;
    output[4*i + 2] = b - (g + b)/2;
    output[4*i + 3] = r + g - 2*(abs(r - g) + b);

}

}
}
""").build()

# Создание буферов памяти на устройстве
mf = cl.mem_flags
input_image_mem_object = cl.Buffer(context, mf.READ_ONLY | mf.COPY_HOST_PTR, hostbuf=input_image)
max_intensity_mem_object = cl.Buffer(context, mf.READ_WRITE | mf.COPY_HOST_PTR, hostbuf=np.zeros(1, dtype=np.int32))
output_image_mem_object = cl.Buffer(context, mf.WRITE_ONLY, output_image.nbytes)
print(output_image.shape)

```

```

nd_range_size_x = num_of_pixels
nd_range_size_y = 1
nd_range_size_z = 1

print(f'ND Range sizes = ({nd_range_size_x}, {nd_range_size_y}, {nd_range_size_z}) Items')

# Установка аргументов ядра OpenCL
grey_scale_program.grey_scale(queue, (nd_range_size_x, nd_range_size_y, nd_range_size_z),
                                  None, input_image_mem_object, output_image_mem_object, max_intensity_mem_object,
                                  np.int32(num_of_pixels))

# Создание буфера на хосте для хранения результата
max_intensity = np.zeros(1, dtype=np.int32)
output_image = np.empty_like(output_image)

# Копирование результата с устройства на хост
cl.enqueue_copy(queue, max_intensity, max_intensity_mem_object)
cl.enqueue_copy(queue, output_image, output_image_mem_object)

output_image = output_image.astype(np.uint8)
output_image = output_image.reshape((image_height, image_width, output_num_of_channels))

io.imshow(output_image)

# Очистка буфера памяти
input_image_mem_object.release()
max_intensity_mem_object.release()
output_image_mem_object.release()

```

```

#### Тестирование
elapsed_time = []
for _ in range(3):

    image_height = image.shape[0]
    image_width = image.shape[1]
    num_of_channels = image.shape[2]
    num_of_pixels = image_height * image_width
    image_size = image_height * image_width * num_of_channels

    output_num_of_channels = 4
    output_image_size = image_height * image_width * output_num_of_channels

    input_image = image.astype(np.int32)
    input_image = input_image.reshape(image_size)

    output_image = np.zeros((image_height, image_width, output_num_of_channels), dtype = np.int32)
    output_image = output_image.reshape(output_image_size)

    start_time = time.time()

    mf = cl.mem_flags
    input_image_mem_object = cl.Buffer(context, mf.READ_ONLY | mf.COPY_HOST_PTR, hostbuf=input_image)
    max_intensity_mem_object = cl.Buffer(context, mf.READ_WRITE | mf.COPY_HOST_PTR,
hostbuf=np.zeros(1, dtype=np.int32))
    output_image_mem_object = cl.Buffer(context, mf.WRITE_ONLY, output_image.nbytes)

    nd_range_size_x = num_of_pixels
    nd_range_size_y = 1
    nd_range_size_z = 1

```

```

# Установка аргументов ядра OpenCL
grey_scale_program.grey_scale(queue, (nd_range_size_x,nd_range_size_y,nd_range_size_z),
                                None, input_image_mem_object, output_image_mem_object, max_intensity_mem_object,
np.int32(num_of_pixels))

# Создание буфера на хосте для хранения результата
max_intensity = np.zeros(1, dtype=np.int32)
output_image = np.empty_like(output_image)

# Копирование результата с устройства на хост
cl.enqueue_copy(queue, max_intensity, max_intensity_mem_object)
cl.enqueue_copy(queue, output_image, output_image_mem_object)

output_image = output_image.astype(np.uint8)
output_image = output_image.reshape((image_height,image_width,output_num_of_channels))

elapsed_time.append(time.time() - start_time)

# Очистка буфера памяти
input_image_mem_object.release()
max_intensity_mem_object.release()
output_image_mem_object.release()

print(f'Среднее время обработки изображения {np.mean(elapsed_time)} секунд')

```

## 2.3 Описание аппаратной базы

Данная программа тестировалась в среде Google Colaboratory, и для получения характеристик аппаратной базы использовался следующий блок кода:

```
def print_device_info():
    print('\n' + '=' * 60 + '\nOpenCL Platforms and Devices')
    for platform in cl.get_platforms():
        print('=' * 60)
        print('Platform - Name: ' + platform.name)
        print('Platform - Vendor: ' + platform.vendor)
        print('Platform - Version: ' + platform.version)
        print('Platform - Profile: ' + platform.profile)
        for device in platform.get_devices():
            print(' ' + '-' * 56)
            print(' Device - Name: ' \
                  + device.name)
            print(' Device - Type: ' \
                  + cl.device_type.to_string(device.type))
            print(' Device - Max Clock Speed: {0} Mhz'\
                  .format(device.max_clock_frequency))
            print(' Device - Compute Units: {0}'\
                  .format(device.max_compute_units))
            print(' Device - Local Memory: {0:.0f} KB'\
                  .format(device.local_mem_size/1024.0))
            print(' Device - Constant Memory: {0:.0f} KB'\
                  .format(device.max_constant_buffer_size/1024.0))
            print(' Device - Global Memory: {0:.0f} GB'\
                  .format(device.global_mem_size/1073741824.0))
            print(' Device - Max Buffer/Image Size: {0:.0f} MB'\
                  .format(device.max_mem_alloc_size/1048576.0))
            print(' Device - Max Work Group Size: {0:.0f}'\
                  .format(device.max_work_group_size))
        print('\n')

print_device_info()
```

```
=====
OpenCL Platforms and Devices
=====
Platform - Name: NVIDIA CUDA
Platform - Vendor: NVIDIA Corporation
Platform - Version: OpenCL 3.0 CUDA 12.0.139
Platform - Profile: FULL_PROFILE
-----
Device - Name: Tesla T4
Device - Type: ALL | GPU
Device - Max Clock Speed: 1590 Mhz
Device - Compute Units: 40
Device - Local Memory: 48 KB
Device - Constant Memory: 64 KB
Device - Global Memory: 15 GB
Device - Max Buffer/Image Size: 3775 MB
Device - Max Work Group Size: 1024
```

Рисунок 4 – Характеристики GPU в Google Colab

Данное задание также выполнялось в Google Colaboratory, поэтому характеристики GPU совпадают.

## 2.4 Примеры входных и выходных изображений

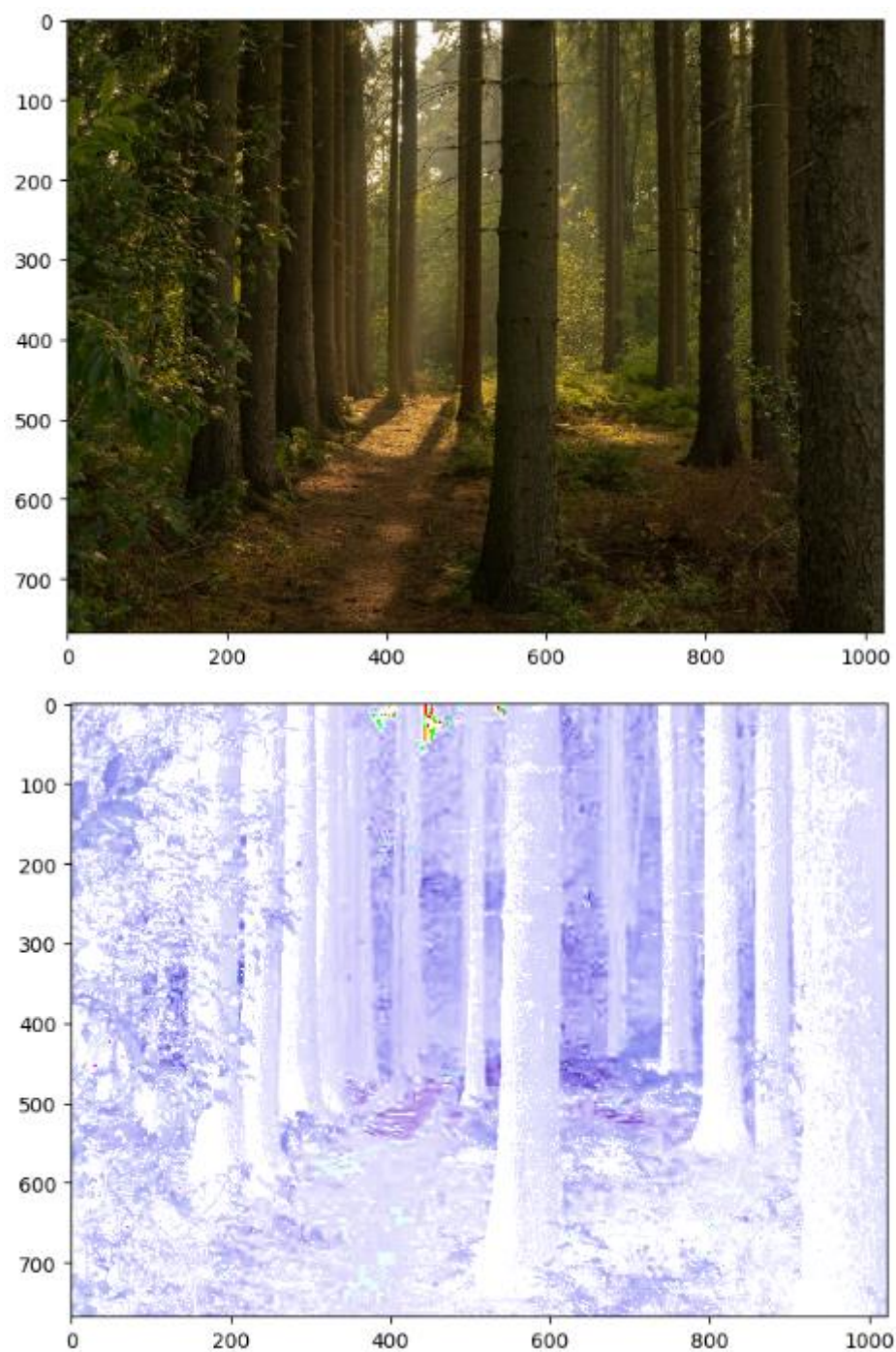


Рисунок 5 – Изображение 768×1024

## 2.5 Производительность программы

Таблица 2 – Среднее время обработки изображений

	768×1024	7680×10240	9600×12800
Среднее время, с	0.013410	1.577120	2.396797

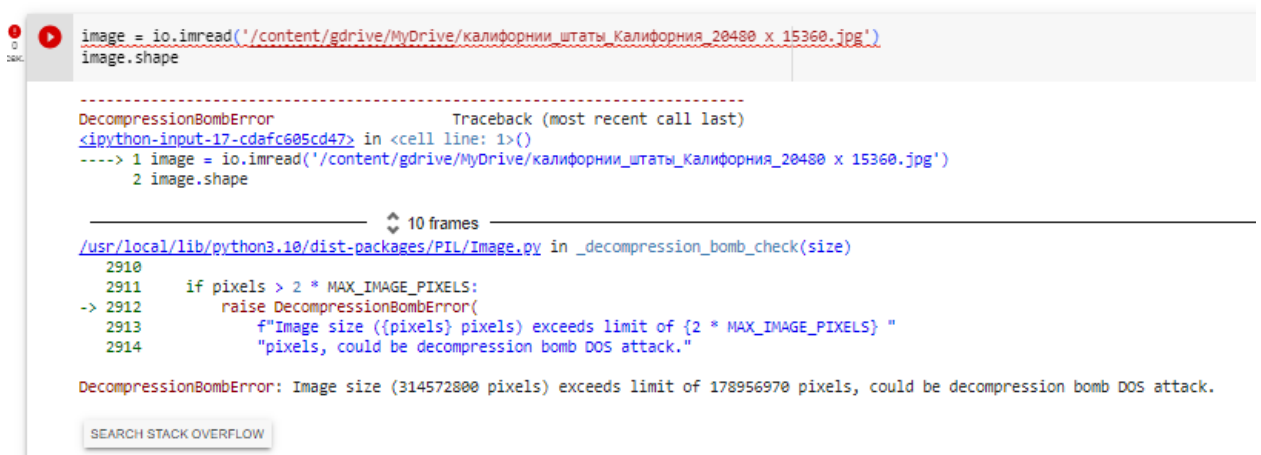


**Выводы:** в ходе выполнения данной лабораторной работы было реализовано две программы: по дилатации и по обработке изображений с использованием инструментов многопоточной обработки на базе GPU. Программа с дилатацией изображения использовала PyCUDA для решения задачи, а программа по обработке изображений применяла PyOpenCL. Обе программы выполнялись в среде Google Colaboratory.

По полученным результатам среднего времени выполнения данных программ, можно сделать вывод, что программа В (обработка изображения на PyOpenCL) выполнялась значительно быстрее, чем программа А (дилатация изображения PyCUDA). Возможно, это связано с тем, что программа А была разделена на два ядра (одно из них выполняло бинаризацию, а другое дилатацию изображения), в отличие от программы В, которая состоит только из одного ядра.

Также стоит отметить, что в ходе выполнения работы не удалось протестировать обработку изображения с разрешением 20480 x 15360 из-за ограничений среды Google Colaboratory.

▼ Изображение с разрешением 20480x15360



```
image = io.imread('/content/gdrive/MyDrive/калифорнии штаты Калифорния_20480 x 15360.jpg')
image.shape

-----
DecompressionBombError                                Traceback (most recent call last)
<ipython-input-17-cdaafc605cd47> in <cell line: 1>()
----> 1 image = io.imread('/content/gdrive/MyDrive/калифорнии штаты Калифорния_20480 x 15360.jpg')
      2 image.shape

-----
10 frames
/usr/local/lib/python3.10/dist-packages/PIL/Image.py in _decompression_bomb_check(size)
    2910
    2911     if pixels > 2 * MAX_IMAGE_PIXELS:
-> 2912         raise DecompressionBombError(
    2913             f"Image size ({pixels} pixels) exceeds limit of {2 * MAX_IMAGE_PIXELS} "
    2914             "pixels, could be decompression bomb DOS attack."
        )
DecompressionBombError: Image size (314572800 pixels) exceeds limit of 178956970 pixels, could be decompression bomb DOS attack.
```

Рисунок 6 – Ограничение Google Colaboratory