

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 №1
по дисциплине «Параллельные и высокопроизводительные вычисления»

Тема работы
Реализация многопоточных вычислений на CPU

Вариант 5

Студент

Группа	ФИО	Подпись	Дата
8BM22	Ямкин Н.Н.		

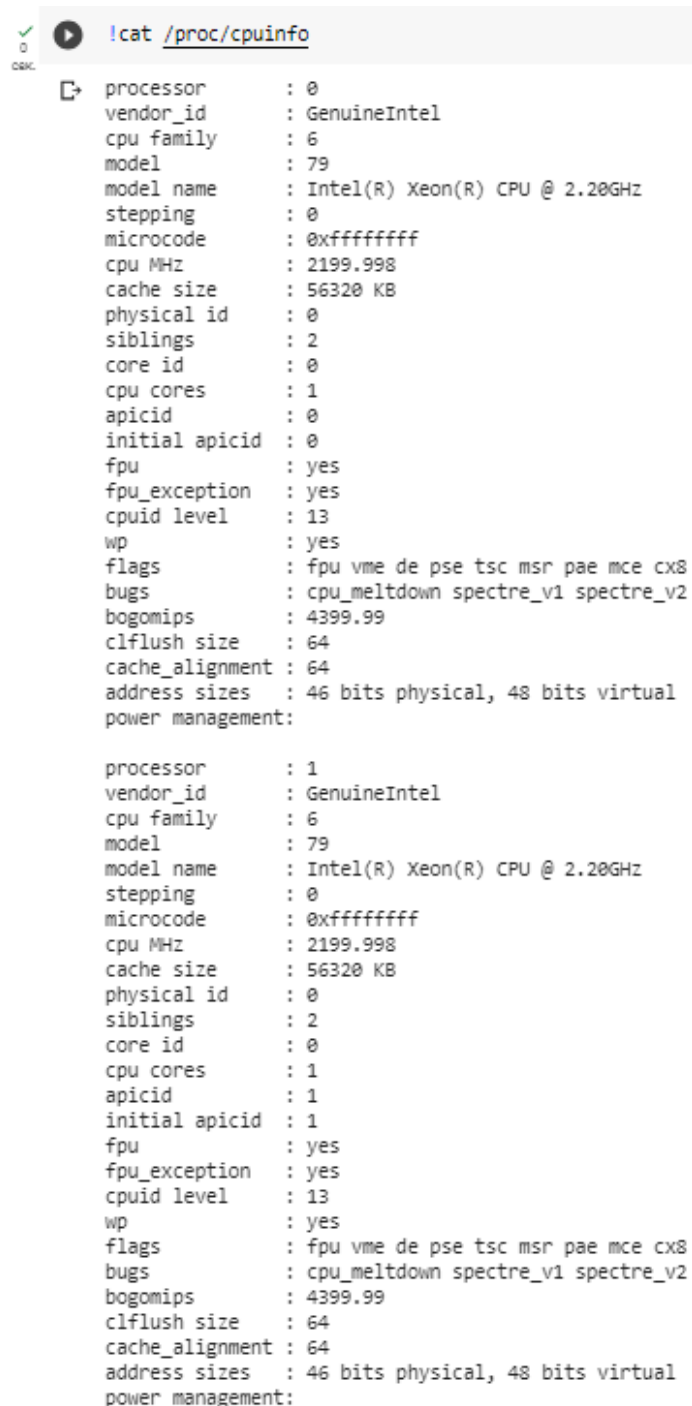
Руководитель

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

Томск – 2023

Ход работы

Данная работа выполнялась на облачной платформе Google Colaboratory. Ниже приведены характеристики выделенных вычислительных ресурсов.



```
!cat /proc/cpuinfo

processor       : 0
vendor_id      : GenuineIntel
cpu family     : 6
model          : 79
model name     : Intel(R) Xeon(R) CPU @ 2.20GHz
stepping       : 0
microcode      : 0xffffffff
cpu MHz        : 2199.998
cache size     : 56320 KB
physical id    : 0
siblings       : 2
core id        : 0
cpu cores      : 1
apicid         : 0
initial apicid : 0
fpu            : yes
fpu_exception  : yes
cpuid level    : 13
wp             : yes
flags          : fpu vme de pse tsc msr pae mce cx8
bugs           : cpu_meltdown spectre_v1 spectre_v2
bogomips       : 4399.99
clflush size   : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id      : GenuineIntel
cpu family     : 6
model          : 79
model name     : Intel(R) Xeon(R) CPU @ 2.20GHz
stepping       : 0
microcode      : 0xffffffff
cpu MHz        : 2199.998
cache size     : 56320 KB
physical id    : 0
siblings       : 2
core id        : 0
cpu cores      : 1
apicid         : 1
initial apicid : 1
fpu            : yes
fpu_exception  : yes
cpuid level    : 13
wp             : yes
flags          : fpu vme de pse tsc msr pae mce cx8
bugs           : cpu_meltdown spectre_v1 spectre_v2
bogomips       : 4399.99
clflush size   : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:
```

Рисунок 1 – Характеристики виртуальной машины

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.

Произвести оценку производительности алгоритма обработки указанных изображений для 2, 4, 6, 8, 10, 12, 14 и 16 потоков на CPU.

1.2 Реализация

Ниже приведен листинг программы.

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import threading
import time
import concurrent.futures
import pandas as pd
from sklearn.cluster import KMeans
from itertools import combinations
pd.options.mode.chained_assignment = None
import warnings
warnings.filterwarnings('ignore')

image_10240_x_7680 = cv2.imread('/content/drive/MyDrive/tropa_les_derevia_10240x7680.jpg')

image_12800_x_9600 = cv2.imread('/content/drive/MyDrive/цветок_лепестки_белый_12800_9600.jpg')

image_20480_x_15360 = cv2.imread('/content/drive/MyDrive/калифорнии_штаты_Калифорния_20480 x 15360.jpg')

def image_processing(picture):
    intensity = (picture[:, :, 0] + picture[:, :, 1] + picture[:, :, 2]) / 3    # вычисляем интенсивность

    # Бинаризация изображения
    thresholdValue = 42    # пороговое значение интенсивности
    maxVal = 1    # на что заменяем, если интенсивность > thresholdValue
    image_binary = cv2.threshold(intensity, thresholdValue, maxVal, cv2.THRESH_BINARY)[1].astype(np.uint8)

    # Инициализируем ядро
    kernel = np.ones((3,3), np.uint8)
```

```

# Дилатация изображения
image_dilation = cv2.dilate(image_binary, kernel, iterations = 1)

return image_dilation    # 1 канальное бинарное изображение + дилатация

def thread_func(process_func, image):
    number_of_threads = [2, 4, 6, 8, 10, 12, 14, 16]
    threads_execution_time = []

    for threads in number_of_threads:
        execute_time = []
        for _ in range(3):
            start_time = time.time()

            with concurrent.futures.ThreadPoolExecutor(max_workers = threads) as executor:
                future = executor.submit(process_func, image)
                dilated_image = future.result()

            execution_time = time.time() - start_time
            execute_time.append(execution_time)
        plt.imshow(f'/content/sample_data/dilated_image_{threads}_threads.jpg', dilated_image, cmap='gray')
        threads_execution_time.append(np.mean(execute_time))
    return threads_execution_time

number_of_threads = [2, 4, 6, 8, 10, 12, 14, 16]

threads_execution_time_image_10240_x_7680 = thread_func(image_processing, image_10240_x_7680)
threads_execution_time_image_12800_x_9600 = thread_func(image_processing, image_12800_x_9600)
threads_execution_time_image_20480_x_15360 = thread_func(image_processing, image_20480_x_15360)

```

1.3 Пример работы программы

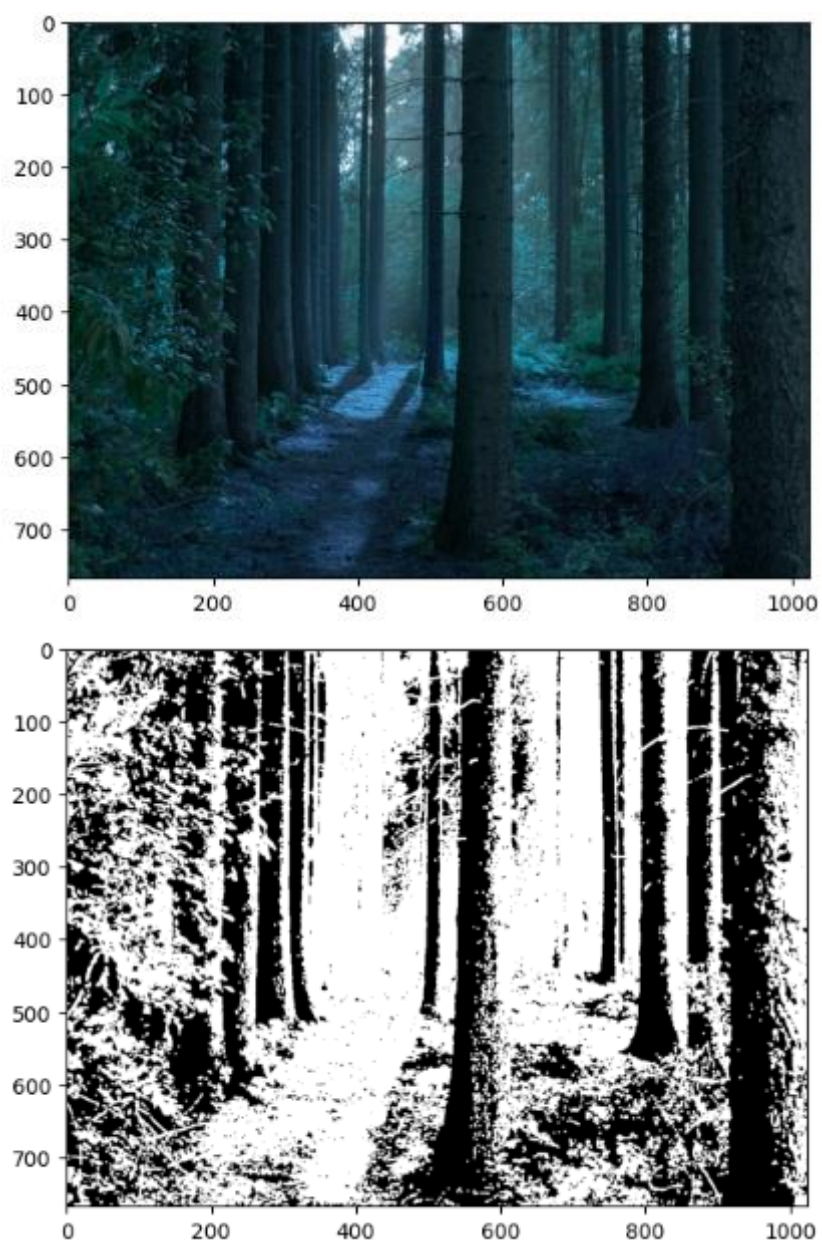


Рисунок 2 – Обработанное изображение 1024×768

1.4 Результаты

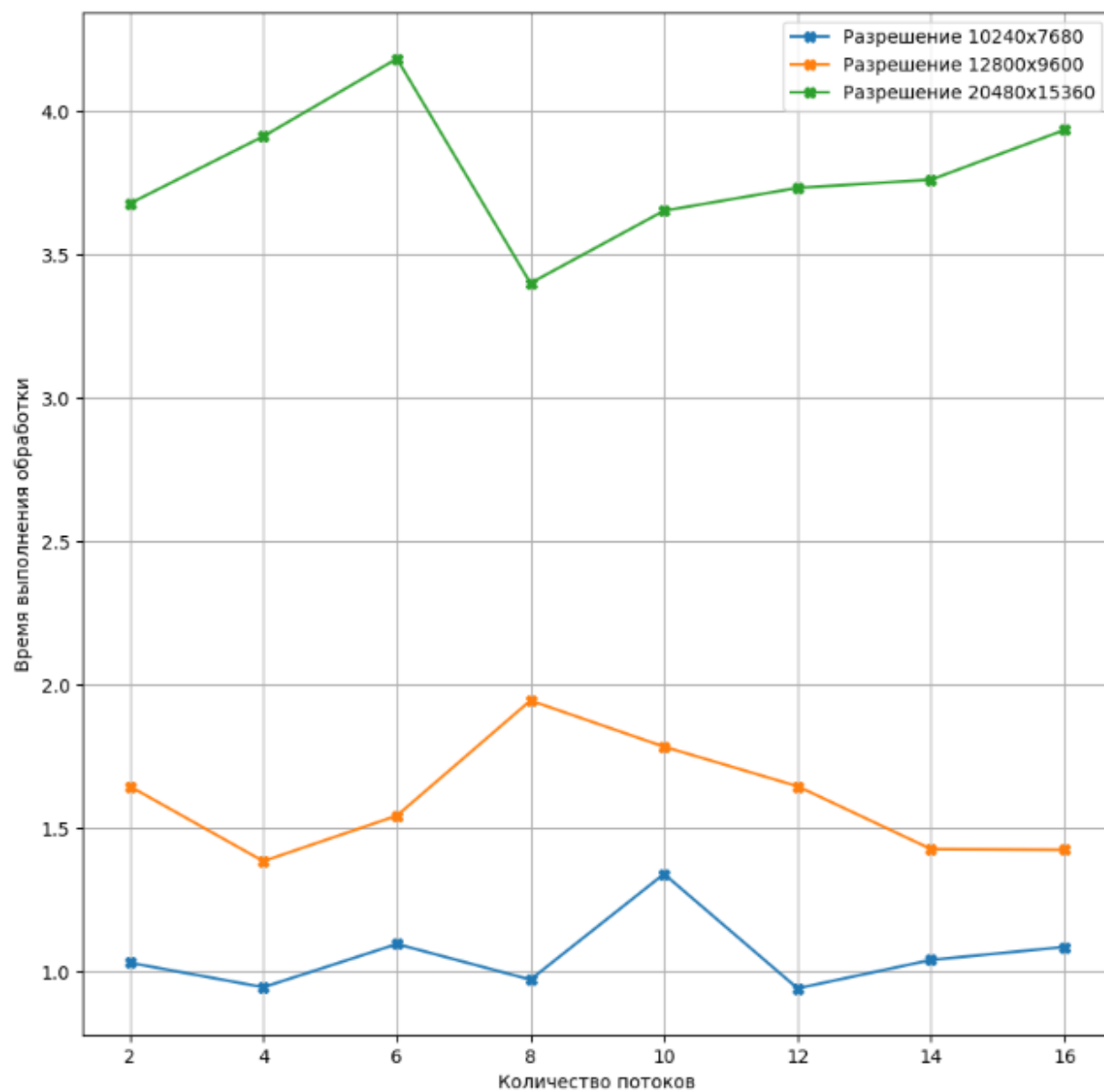


Рисунок 3 – Результаты работы программы

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.

Выполнить инвертирование цветов пикселей (новое значение цветового канала соответствует значению 255 – старое значение цветового канала). Выполнить свертку с фильтром, увеличивающим контраст, с каждым из цветовых каналов изображения. Ядро преобразования можно найти по ссылке: [<https://docs.gimp.org/2.8/ru/plugin-convmatrix.html>]. Сохранить результат в файл.

Произвести оценку производительности алгоритма обработки указанных изображений для 2, 4, 6, 8, 10, 12, 14 и 16 потоков на CPU.

2.2 Реализация

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import threading
import time
import concurrent.futures
import pandas as pd
from sklearn.cluster import KMeans
from itertools import combinations
pd.options.mode.chained_assignment = None
import warnings
warnings.filterwarnings('ignore')

def conv_processing(image):
    # Инвертируем изображение
    invert_image = 255 - image

    # Ядро обнаружения границ
    kernel = np.array([[ -1,  -1,  -1],
                       [ -1,  8,  -1],
                       [ -1, -1,  -1]])

    # Применяем фильтр
    result_image = cv2.filter2D(src=invert_image, ddepth=-1, kernel=kernel)
    return result_image

threads_execution_time_image_10240_x_7680 = thread_func(conv_processing, image_10240_x_7680)
threads_execution_time_image_12800_x_9600 = thread_func(conv_processing, image_12800_x_9600)
threads_execution_time_image_20480_x_15360 = thread_func(conv_processing, image_20480_x_15360)
```

2.3 Примеры входных и выходных изображений

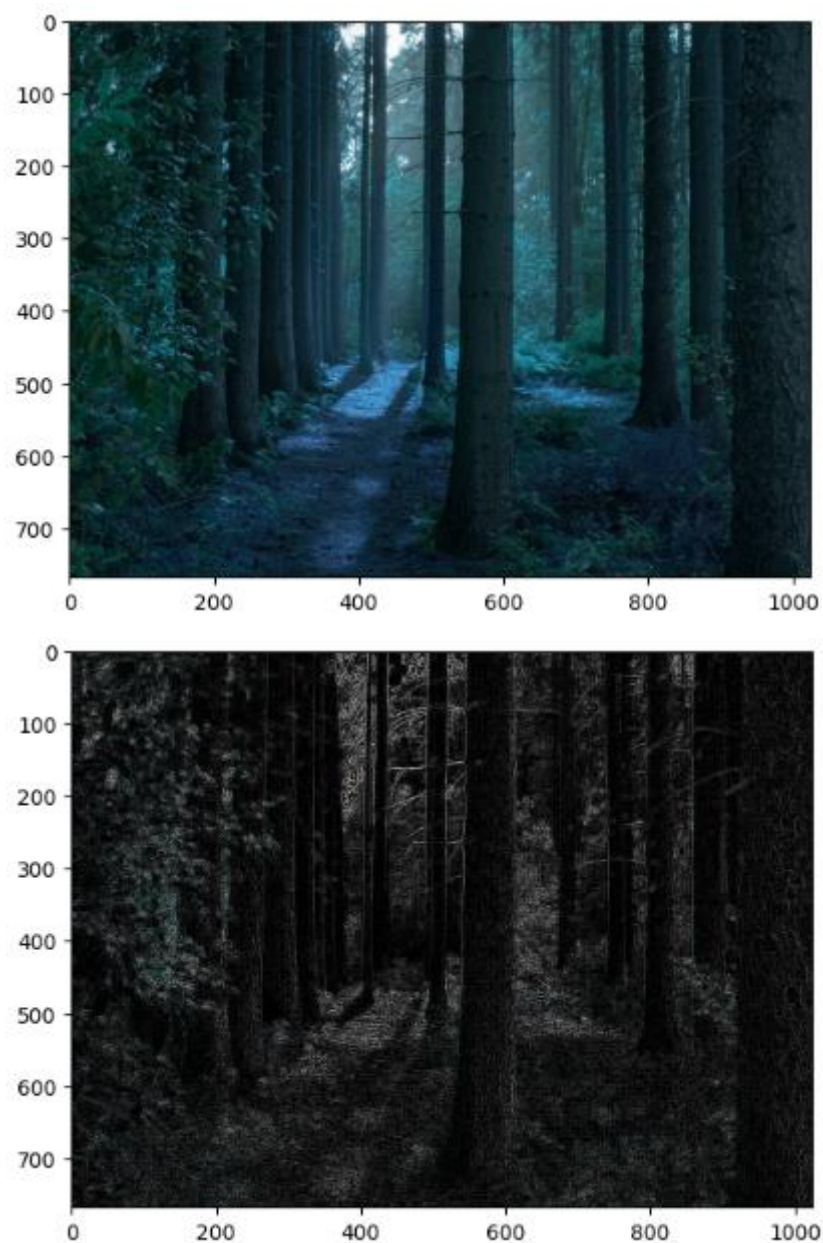


Рисунок 4 – Обработанное изображение 1024×768

2.4 Результаты

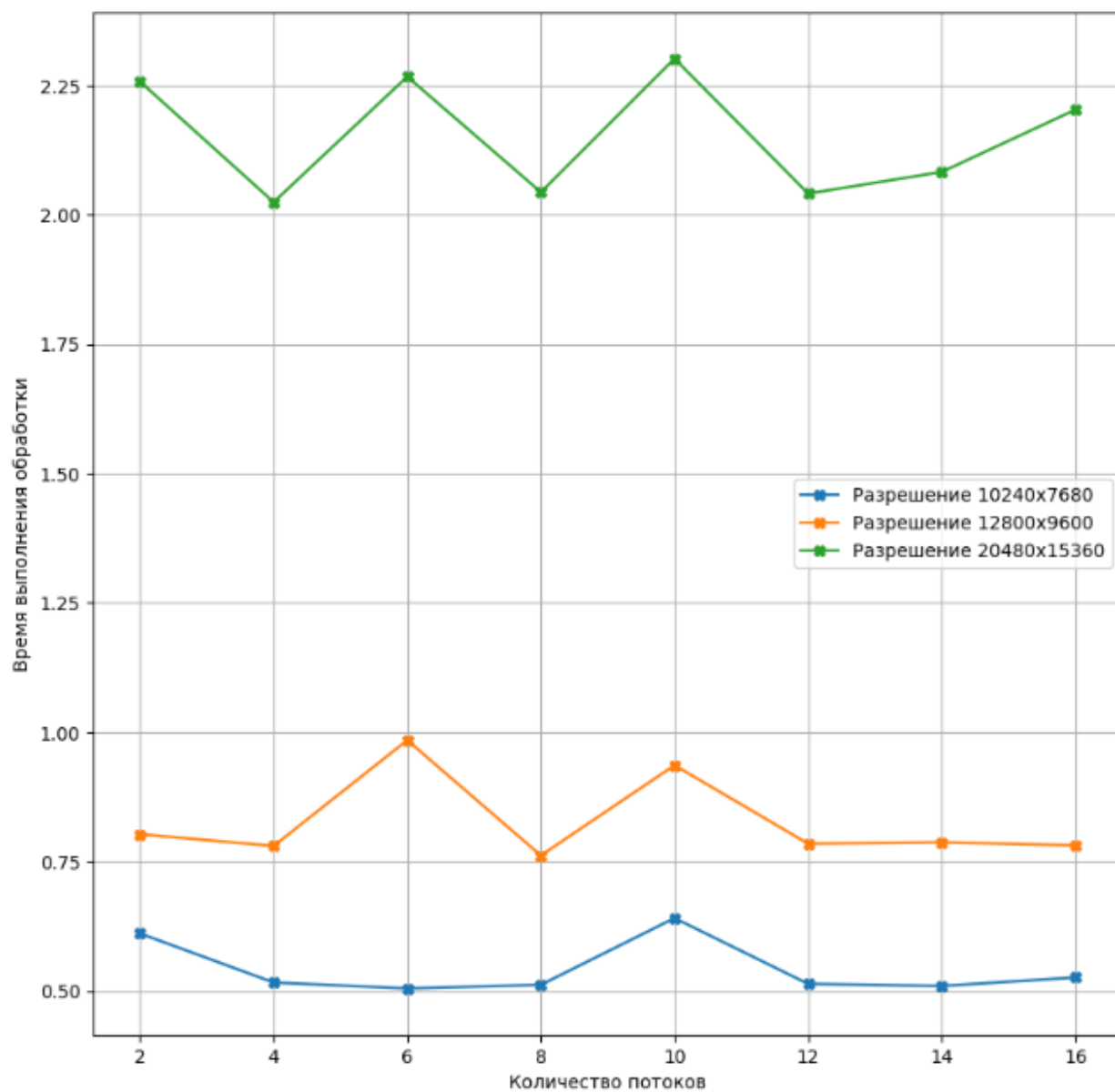


Рисунок 5 – Результаты работы программы

3 Программа С

3.1 Задание

Загрузка набора данных «BD-Patients.csv». Для исследования взять столбцы «HCT_mean», «Urine_mean». Выполнить кластеризацию загруженных данных с помощью метода К-средних (предварительно отмасштабировать признаки в диапазон $[0, 1]$). Количество кластеров $K = 3, 4, 5$.

Написать программу (функцию), реализующую расчет индекса Maulik-Bandopadhyay для оценки кластерной структуры с использованием многопоточных вычислений. Оценить качество решения с помощью указанного метода. Вывести значения центров кластеров для и визуализировать кластерное решение. Произвести оценку производительности алгоритма для 2, 4, 6, 8, 10, 12, 14 и 16 потоков на CPU для расчета индекса для решения, полученного для 1000, 3000 и 5000 векторов.

3.2 Реализация

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import threading
import time
import concurrent.futures
import pandas as pd
from sklearn.cluster import KMeans
from itertools import combinations
pd.options.mode.chained_assignment = None
import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('/content/drive/MyDrive/BD-Patients.csv')[['HCT_mean', 'Urine_mean']]

# Приводим значения в диапазон от 0 до 1
df['HCT_mean'] = df['HCT_mean'] / df['HCT_mean'].max()
df['Urine_mean'] = df['Urine_mean'] / df['Urine_mean'].max()
# Заполняем пропуски средним значением
df['HCT_mean'][df['HCT_mean'].isnull()] = df['HCT_mean'].mean()
df['Urine_mean'][df['Urine_mean'].isnull()] = df['Urine_mean'].mean()

kmeans_global_center = KMeans(n_clusters=1).fit(df)
global global_center
global_center = kmeans_global_center.cluster_centers_[0][:2]

df['cluster'] = kmeans_global_center.predict(df)
df_1000 = df[:1000]
df_3000 = df[:3000]
df_5000 = df[:5000]
```

```

def graphic(kmeans, data):
    fig = plt.figure(figsize=(10, 10))
    plt.subplot(1, 1, 1)
    cluster_centers_coord = kmeans.cluster_centers_
    for cluster_number in range(len(cluster_centers_coord)):
        plt.scatter(data['HCT_mean'][data['cluster'] == cluster_number], data['Urine_mean'][data['cluster'] ==
cluster_number])
    plt.xlabel('HCT_mean')
    plt.ylabel('Urine_mean')
    return plt.scatter(cluster_centers_coord[:,0], cluster_centers_coord[:,1], marker = '.', s = 100, color = 'black')

def MB_index_calculating(data):
    clusters_number_list = [3,4,5]
    MB_index_list = []

    for clusters in clusters_number_list:
        kmeans = KMeans(n_clusters=clusters).fit(data)
        data['cluster'] = kmeans.predict(data)

        distance_between_cluster_centers = []
        inner_cluster_distances_sum = 0
        global_distance_sum = 0

        for cluster_center_1, cluster_center_2 in combinations(kmeans.cluster_centers_[0, :2], 2):
            distance_between_cluster_centers.append(np.linalg.norm(cluster_center_1 - cluster_center_2))
        max_distance_between_cluster_centers = max(distance_between_cluster_centers)

        for cluster, cluster_center in enumerate(kmeans.cluster_centers_[0, :2]):
            data_cluster = data[data['cluster'] == cluster]

            for index, row in data_cluster.iterrows():

```

```

        x_vector = (row['HCT_mean'], row['Urine_mean'])
        inner_cluster_distances_sum += np.linalg.norm(cluster_center - x_vector)
        global_distance_sum += np.linalg.norm(global_center - x_vector)

    MB_index_list.append(np.square(1/clusters * global_distance_sum/inner_cluster_distances_sum *
max_distance_between_cluster_centers))
    # data = data.drop(['cluster'])
    return(MB_index_list)

def thread_func(process_func, data):
    number_of_threads = [2,4,6,8,10,12,14,16]
    # number_of_threads = [2]
    threads_execution_time = []

    for threads in number_of_threads:
        execute_time = []
        for _ in range(3):
            start_time = time.time()

            with concurrent.futures.ThreadPoolExecutor(max_workers = threads) as executor:
                future = executor.submit(process_func, data)
                processed_data = future.result()

            execution_time = time.time() - start_time
            execute_time.append(execution_time)
        threads_execution_time.append(np.mean(execute_time))
    return threads_execution_time

threads_execution_time_df_1000 = thread_func(MB_index_calculating, df_1000)
threads_execution_time_df_3000 = thread_func(MB_index_calculating, df_3000)
threads_execution_time_df_5000 = thread_func(MB_index_calculating, df_5000)

```

3.3 Визуализация кластерного решения

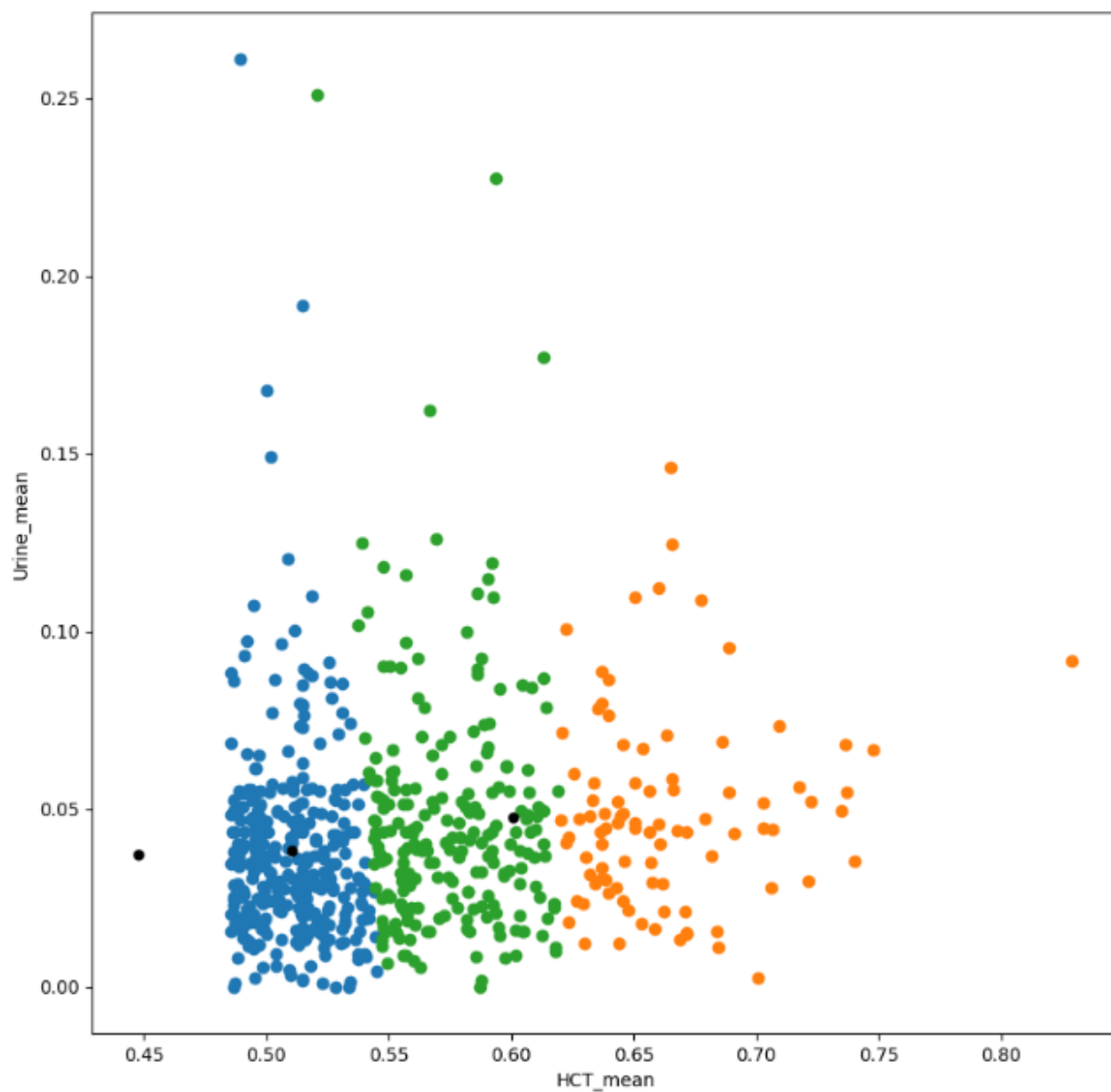


Рисунок 6 – Результат кластеризации при $K = 3$

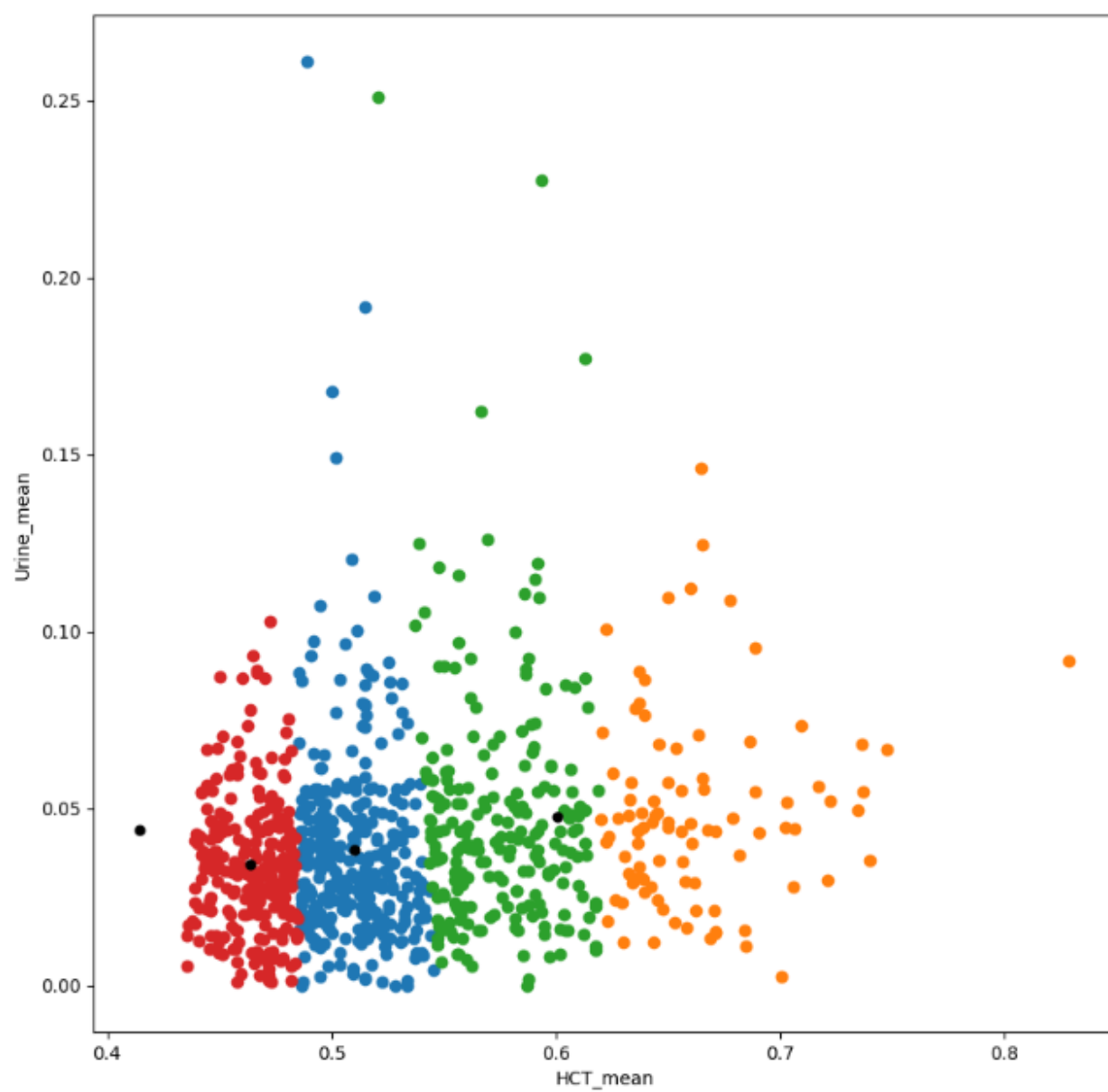


Рисунок 7 – Результат кластеризации при $K=4$

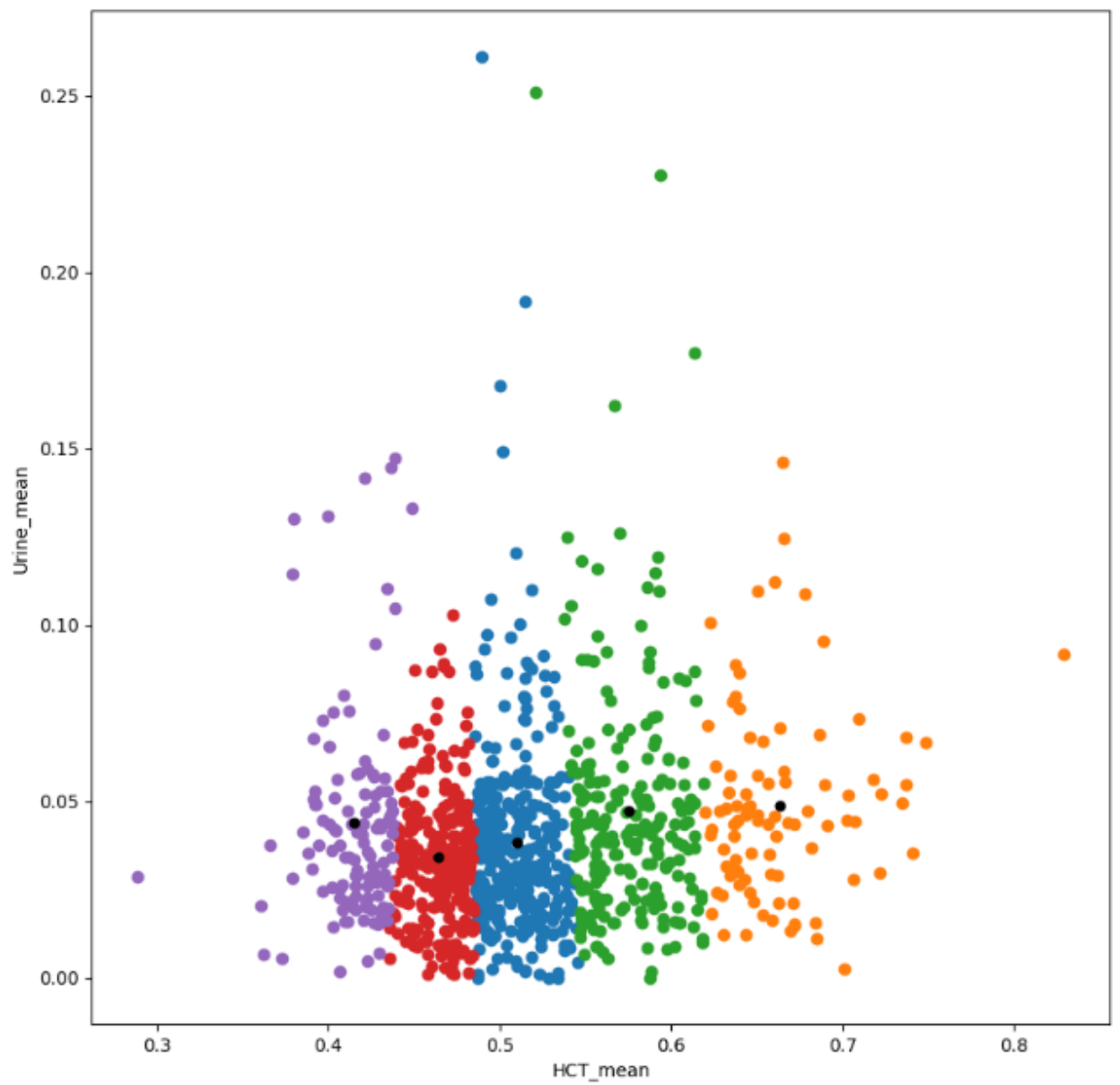


Рисунок 8 – Результат кластеризации при $K = 5$

3.4 Результаты

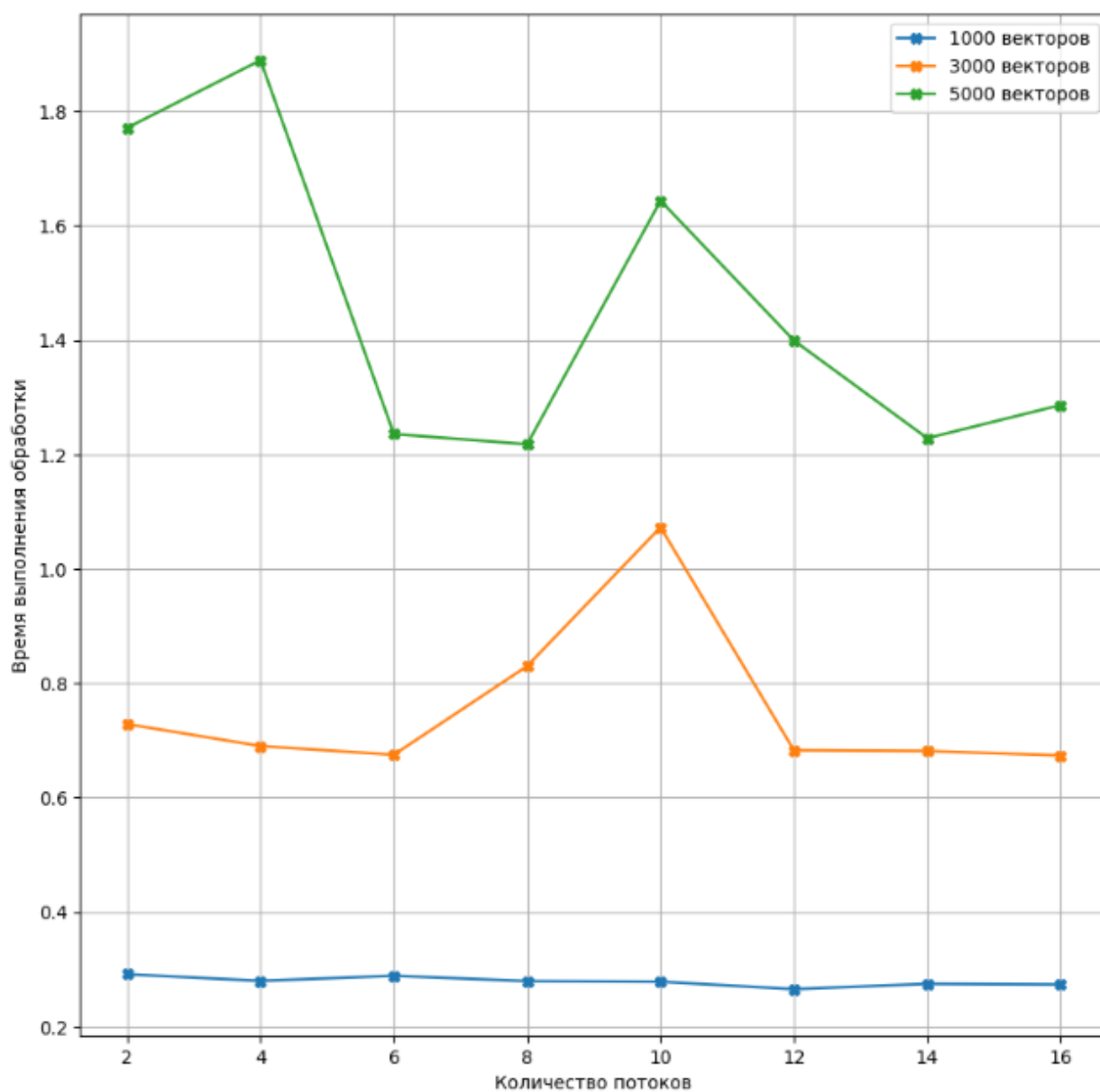


Рисунок 9 – Результат выполнения программы

Выводы: в ходе выполнения данной лабораторной работы было реализовано две программы по обработке изображений и программа для решения задачи кластеризации. Многопоточная обработка данных выполнялась в облачном сервисе Google Colab. По графикам зависимости времени выполнения программы от количества потоков видно, что время увеличивается с увеличением размера обрабатываемых данных и слабо зависит от количества потоков на CPU (разница между самым быстрым и самым долгим выполнением программы на одном и том же наборе данных составляет приблизительно 0.5 секунд). Данный результат связан с ограничением языка Python GIL (Global Interpreter Lock), которое позволяет выполняться только одному конкретному потоку в любой момент времени.