МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №3
по курсу «Программирование графических процессоров»

Классификация и кластеризация изображений на GPU.

Выполнил: О.А. Мезенин
Группа: 8О-406Б
Преподаватель: А.Ю. Морозов

Москва, 2024

Условие

Цель работы: Научиться использовать GPU для классификации и кластеризации изображений. Использование константной памяти и одномерной сетки потоков.

Вариант 2. Метод расстояния Махаланобиса.

Формат изображений соответствует формату описанному в лабораторной работе 2. Во всех вариантах, в результирующем изображении, на месте альфа-канала должен быть записан номер класса(кластера) к которому был отнесен соответствующий пиксель. Если пиксель можно отнести к нескольким классам, то выбирается класс с наименьшим номером.

На первой строке задается путь к исходному изображению, на второй, путь к конечному изображению. На следующей строке, число nc -- количество классов. Далее идут nc строчек описывающих каждый класс. В начале j-ой строки задается число np_j -- количество пикселей в выборке, за ним следуют np_j пар чисел -- координаты пикселей выборки. $nc \le 32$, $np_j \le 2^{19}$, $w*h \le 4*10^8$.

Для некоторого пикселя p, номер класса jc определяется следующим образом:

$$jc = arg\,max_j\left[-(p - avg_j)^T * cov_j^{-1} * (p - avg_j)\right]$$

Программное и аппаратное обеспечение

Характеристики графического процессора:

- Compute capability: 8.9
- Name: NVIDIA GeForce RTX 4070 SUPER
- Total Global Memory: 12584550400
- Shared memory per block: 49152
- Registers per block: 65536
- Warp size: 32
- Max threads per block: (1024, 1024, 64)
- Max block: (2147483647, 65535, 65535)
- Total constant memory: 65536
- Multiprocessors count: 56

Характеристики центрального процессора:

- Version: AMD Ryzen 7 7700
- Max Speed: 5389 MHz
- Core Count: 8
- Thread Count: 16

Характеристики оперативной памяти:

- Size: 2x16 GB
- Type: DDR5
- Speed: 4800 MT/s

Характеристики жесткого диска:

- Device Model: ADATA SU635
- User Capacity: 480 103 981 056 bytes [480 GB]

- Sector Size: 512 bytes logical/physical
- SATA Version is: SATA 3.2, 6.0 Gb/s (current: 6.0 Gb/s)

Программное обеспечение:

- OS: Linux
- Kernel: 6.10.13-3-MANJARO
- Текстовый редактор: Kate
- Компилятор: nvcc V12.4.131

Метод решения

На хосте вычислим оценку вектора средних и ковариационной матрицы по заданным формулам:

$$avg_j = \frac{1}{np_j} \sum_{i=1}^{np_j} ps_i^j$$

$$cov_j = \frac{1}{np_j - 1} \sum_{i=1}^{np_j} (ps_i^j - avg_j) * (ps_i^j - avg_j)^T$$

где $ps_i^j = (r_i^j \; g_i^j \; b_i^j)^T$ -- i-ый пиксель из j-ой выборки.

Также вычислим обратную ковариационную матрицу, при этом сделав её одномерной. Заведём переменные *avg_dev* и *cov_inv_dev* с квалификатором *__constant__*, которые будут храниться в константной памяти GPU. Скопируем в них вычисленные оценку вектора средних и обратную ковариационную матрицу.
В ядре опишем данную в варианте формулу. Номер класса, к которому был отнесен соответствующий пиксель, будем записывать на месте альфа-канала.

Описание программы

Вся программа описана в одном файле main.cu. Есть следующие функции:

- int main() — входная точка программы, которая считывает данные, вычисляет вектор средних и обратную ковариационную матрицу, копирует их в константную память, запускает ядро, и записывает результат;
- void readData(std::string& filename, int& w, int& h, uchar4** data) — открывает файл filename, считывает оттуда размеры изображения и записывает их в w и h, после чего считывает данные изображения и записывает их в data;
- void writeData(std::string& filename, int w, int h, uchar4* data) — записывает изображение data размером w на h в файл filename;
- float3* calculateAvgVector(uchar4 *data, int width, std::vector<std::vector<int2>>& coors) — вычисляет вектор средних.
- float3x3* calculateCovMatrix(uchar4 *data, int width, std::vector<std::vector<int2>>& coors, float3* avg) — вычисляет ковариационную матрицу.
- float calculateDet(float3x3 m) — вычисляет дискриминант матрицы 3x3.
- float3* calculateCovInvVector(float3x3* cov, int nc) — вычисляет обратную ковариационную одномерную матрицу размера 3*nc.

- __device__ unsigned char calculateClassUsingMahalanobisDistanceMethod(int nc, uchar4 p) — для пикселя p определяет номер класса, к которому он относится, используя метод расстояния Махаланобиса.
- __global__ void kernel(uchar4 *data, int width, int height, int nc) — ядро проходится по одномерной сетке потоков и вычисляет для каждого пикселя номер класса.

Результаты

Изображения для тестов будут следующих размеров: 512x512, 2048x2048, 8192x8192. Количество классов равно 4.

Далее приведены замеры времени работы ядер с тремя конфигурациями, а также замеры работы без использования технологий CUDA (на CPU):

1. Конфигурация <<<1, 32>>>

WxH	Время (в мс)
512x512	131.63
2048x2048	1928.39
8192x8192	30986.9

2. Конфигурация <<<32, 32>>>

WxH	Время (в мс)
512x512	4.04349
2048x2048	64.2458
8192x8192	988.367

3. Конфигурация <<<1024, 1024>>>

WxH	Время (в мс)
512x512	2.04902
2048x2048	32.1424
8192x8192	500.222

4. CPU

WxH	Время (в мс)
512x512	8.965
2048x2048	143.57
8192x8192	2100.35

Ниже приведены примеры работы программы. Пиксели разных цветов относятся к разным классам.

Рис 1 — Исходное изображение примера 1

Рис 2 — Выходное изображение примера 1

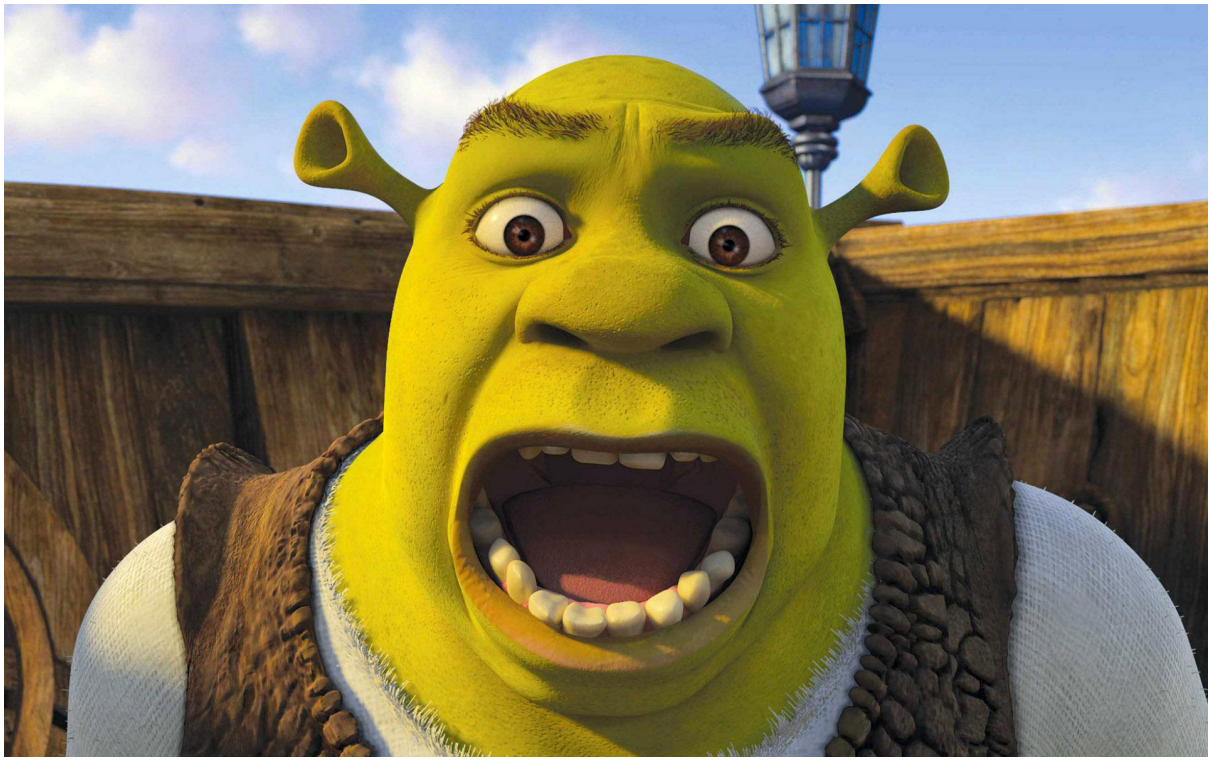

Рис 3 — Исходное изображение примера 2

Рис 4 — Выходное изображение примера 2

Выводы

В ходе лабораторной работы ознакомился с понятием классификации и кластеризации изображений. Также узнал про константную память на GPU и использовал её в лабораторной работе. Константная память используется для хранения данных, которые остаются неизменными во время выполнения программы. Она обеспечивает более быстрый доступ к этим данным по сравнению с основной памятью.

Реализовал метод расстояния Махаланобиса для классификации пикселей. Провёл сравнительный анализ работы алгоритма с использованием технологий CUDA и без них.

Из результатов сравнения можно увидеть, что конфигурация с самым маленьким количеством блоков и потоков отстаёт по времени на порядки от остальных. При этом алгоритм на CPU отстаёт от двух самых больших конфигураций только в разы, хотя в предыдущих лабораторных работах алгоритмы на CPU были заметно медленнее. Возможно, в этот раз разница была бы заметнее на ещё более больших тестах.