МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

**Лабораторная работа №3**
**по курсу «Программирование графических процессоров»**

# Классификация и кластеризация изображений на GPU.

Выполнил: Н.И. Забарин
Группа: 8О-408Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2017

## Условие

1. **Цель работы:**
   Научиться использовать GPU для классификации и кластеризации изображений. Использование *константной памяти*.
2. **Вариант задания:**
   Вариант 2. Метод расстояния Махаланобиса.

## Программное и аппаратное обеспечение

**Спецификации GPU**

| | |
|---|---|
| Name: | GeForce GT 620M |
| Compute capability: | 2.1 |
| Warp size: | 32 |
| Max threads per block: | 1024 |
| Clock rate: | 1250000 |
| Multiprocessor count: | 2 |
| Max threads dim: | 1024 1024 64 |
| Max grid size: | 65535 65535 65535 |

**Спецификации видеопамяти**

| | |
|---|---|
| Total global memory: | 1024 MB |
| Shared memory per block: | 48 KB |
| Registers per block: | 32 KB |
| Total constant memory: | 64 KB |

**Спецификации CPU**

| | |
|---|---|
| Процессор | Intel Core i5-3317U |
| Ядер | 4 |
| Базовая частота | 1.7 GHz |

**Спецификация оперативной памяти**

| | |
|---|---|
| Объем памяти | 10 Гб |
| Частота | 1600 МГц |

**Спецификация жесткого диска**

| | |
|---|---|
| Тип | SSD |
| Интерфейс | M.2 |
| Объём | 240Gb |

**Спецификация программного обеспечения**

| | |
|---|---|
| CUDA Toolkit | 7.5 |
| OS | Ubuntu 16.10 |
| IDE | Vim |
| Compiler | nvcc V7.5.17 |

## Метод решения

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

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

Оценка вектора средних и ковариационной матрицы:

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

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

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

,

npj – количество пикселей в выборке.

Обратная матрица матрицы A ищется как

$$A^{-1} = \frac{\mathrm{adj}(A)}{\det(A)}$$

где adj(A) — присоединенная матрица.

$$C^* = \begin{pmatrix} A_{11} & A_{21} & \cdots & A_{n1} \\ A_{12} & A_{22} & \cdots & A_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ A_{1n} & A_{2n} & \cdots & A_{nn} \end{pmatrix}$$

Исходная матрица:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

Где:

- $C^*$ — присоединённая(союзная, взаимная) матрица;
- $A_{ij}$ — алгебраические дополнения исходной матрицы;
- $a_{ij}$ — элементы исходной матрицы.

**Алгебраическим дополнением** элемента $a_{ij}$ матрицы $A$ называется число

$$A_{ij} = (-1)^{i+j} M_{ij},$$

где $M_{ij}$ — дополнительный минор.

## Описание программы

В программе имеется одно ядро
__global__ void kernel(uchar4 *data, uint32_t w, uint32_t h, int cnum).

В нем проходятся все пиксели, соотнесенные с данным потоком, и у каждого пикселя в компоненту альфа канала записывается номер класса внутри функции
__device__ void classify(pixel &p, int cnum).

В функции classify для каждого класса вызывается функция
__device__ double mahalanobis(const pixel p, int ci),
которая рассчитывается квадрат расстояния Махаланобиса с обратным знаком от пикселя до заданного класса.

Выбирается класс, для которого функция выдает наибольшее значение.

Для каждого класса в константной памяти GPU объявлены два массива
__constant__ double GPU_AVG[32][3];
__constant__ double GPU_INVERT_COV[32][3][3];
хранящие векторы средних и обратные ковариационные матрицы.

## Результаты

Количество классов - 8

| Размер изображения | Параметры ядра | Время |
|---|---|---|
| 400x326 | <<<dim3(8, 8), dim3(8, 8)>>> | 1.243392 |
| 400x326 | <<<dim3(16, 16), dim3(16, 16)>>> | 1.236064 |
| | | |
| 1300x975 | <<<dim3(8, 8), dim3(8, 8)>>> | 10.477120 |
| 1300x975 | <<<dim3(16, 16), dim3(16, 16)>>> | 10.460032 |
| | | |
| 6000x4000 | <<<dim3(8, 8), dim3(8, 8)>>> | 197.120636 |
| 6000x4000 | <<<dim3(16, 16), dim3(16, 16)>>> | 193.203613 |
| 6000x4000 | <<<dim3(100, 100), dim3(32, 32)>>> | 195.232864 |

## Выводы

При распараллеливании задач на CUDA следует учитывать возможность помещения каких-либо объектов в константную память. Помещение каких-либо общих для всех потоков данных, которые не будут меняться, в константную память, может увеличить скорость работы приложения за счет увеличения скорости обмена данными с памятью.

Задачи метрической классификации могут быть успешно распараллелены, т.к. классификация каждого объекта производится независимо.