МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №3
по курсу «Программирование графических процессоров»

Классификация и кластеризация изображений на *GPU*

Выполнил: Днепров И. С.
Группа: 8О-407Б-17
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2021

Условие

Цель работы:

Научиться использовать *GPU* для классификации и кластеризации изображений.
Использование константной памяти.

Формат изображений

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.

Вариант 2. Метод расстояния Махаланобиса

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.

$nc \le 32, np_j \le 2^{19},\ wh \le 4 \cdot 10^8$.

Оценка вектора средних:

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

оценка ковариационной матрицы:

$$cov_j = \frac{1}{np_j - 1} \sum_{i=1}^{np_j} \left(ps_i^j - avg_j\right) \cdot \left(ps_i^j - avg_j\right)^T,$$

$где\ ps_i^j = \begin{pmatrix} r_i^j \\ g_i^j \\ b_i^j \end{pmatrix}$ – i-й пиксель j-й выборки.

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

$$jc = \arg \max_{1 \le j \le nc} \left[-\left(p - avg_j\right)^T \cdot cov_j^{-1} \cdot \left(p - avg_j\right) \right] .$$

Пример:

Входной файл	hex: in.data	hex: out.data
in.data out.data 2 4 1 2 1 0 2 2 2 1 4 0 0 0 1 1 1 2 0	03000000 03000000 A2DF4C00 F7C9FE00 9ED84500 B4E85300 99D14D00 92DD5600 A9E04C00 F7D1FA00 D4D0E900	03000000 03000000 A2DF4C01 F7C9FE00 9ED84501 B4E85301 99D14D01 92DD5600 A9E04C01 F7D1FA00 D4D0E900

Входной файл	*hex: out.data*

in.data out.data 5 4 5 0 0 2 6 1 1 1 6 2 0 7 1 1 0 1 2 6 0 4 0 4 3 0 3 1 0 1 0 0 4 0 3 6 2 5 2 7 2 9 6 4 5 1 7 0 2 1 2 3 4 1 1 5 3 3 2 6	08000000 08000000 D2E27502 CFF65201 D3ED5701 D6E76902 C8F35B01 8E168200 CFF45001 AE977604 D3DC7102 7D1E7B00 AB9A8004 D9E58602 AB967E04 AE9D8004 87058200 D0F95B01 74148000 D0F55901 86136C00 85077400 D6E27702 D3609F03 D1609F03 CC5EA103 CC739D03 7C127F00 AA988804 AFA07D04 D0E37702 7D117A00 D6EB5901 D6E37C02 C9F85701 D655A103 D7EA7402 93127D00 D35BA403 D4DD7902 B0A18404 D6DE7502 D765A900 AD928404 D0D87C02 D7E97F02 CD509E00 CAF85201 CFF75601 CEF45E01 D0E86902 D1D17F02 AD928104 AFA18304 D4DB5C02 88077D00 C6F75701 7D127D00 A99A8E04 C8609E03 D15DA503 AB957E04 AE9A8004 79218100 D065A103 A99E9A04

Программное и аппаратное обеспечение

GPU

Название: GeForce GT 545
Размер глобальной памяти: 3150381056
Размер константной памяти : 65536
Размер разделяемой памяти: 49152
Регистров на блок: 32768
Максимум потоков на блок: 1024
Размер варпа: 32
Максимальные размеры блока: 1024 x 1024 x 64
Максимальные размеры сетки: 65535 x 65535 x 65535
Количество мультипроцессоров : 3

CPU

Название: Intel Core i7-3770

Частота: 3.40GHz

Размер кеша: 8192 KB

Количество ядер: 4

Количество потоков: 8

MEM

Размер: 15 GB

Тип: ddr3

Прочее

OS: Linux Ubuntu 16.04.6
Редактор: Atom

Метод решения

1. Считываем размер входного изображения, выделяем нужный объём памяти на *CPU* и *GPU*. Заводим статические массивы *avg* и *cov* на *CPU* и такие же – на *GPU* с квалификатором __constant__. В этих массивах будут храниться avg_j и cov_j^{-1} соотвественно.
2. Считываем пиксели изображения из входного файла в массив.
3. Считываем индексы элементов выборки, одновременно высчитывая avg_j и cov_j^{-1}(обращение матрицы выполняем с помощью *LUP*-разложения и решения соответствующих линейных уравнений) на *CPU* .
4. Копируем avg_j и cov_j^{-1} в константную память на *GPU*, а также само изображение в глобальную память.
5. Запускаем ядро, в котором для каждого пикселя p находим номер его класса:$jc = \arg\max_{1 \le j \le nc} \left[-\left(p - avg_j\right)^T \cdot cov_j^{-1} \cdot \left(p - avg_j\right) \right]$

Описание программы

Прототип	Тип	Описание
`#define NC_MAX`	Макрос константа	Максимальное количество классов
`double dot(const double *A, const double *a, const double *b, int n)`	Функция	Скалярное произведение $a^T \cdot A \cdot b$
`void mat_mul(const double *A, const double *B, double *C, int m, int n, int l, double alpha=1, double beta=0)`	Функция	$C = \alpha AB + \beta C$
`void mat_mul_C(const double *A, double *B, double c, int m, int n)`	Функция	$B = cA$
`void mat_sum(const double *A, const double *B, double *C, int m, int n, double alpha=1, double beta=1)`	Функция	$C = \alpha A + \beta B$
`void mat_set(double *A, int n, double alpha=0, double beta=0)`	Функция	$A = \|\|a_{ij}\|\|$, где $a_{ij} = \begin{cases} \alpha, i = j \\ \beta, i \neq j \end{cases}$
`void mat_swap_rows(double *A, int n, int i1, int i2)`	Функция	Перестановка местами строк матрицы
`void mat_transpose(const double *A, double *A_t, int n)`	Функция	Транспонирование матрицы
`bool LUP(double *A, int n, int *pi, double eps=1e-10)`	Функция	Поиск L, U и массива перестановок π (P) так что $PA = LU$
`void LUP_solve(const double *LU, const int *pi, const double *b, int n, double *x, double *work)`	Функция	Решение линейной системы $LUx = Pb$
`void LUP_inv_mat(double *A, double *A_inv, int n, double *work, int *iwork, double eps=1e-10)`	Функция	Поиск обратной матрицы A^{-1}
`__global__ void kernel(int nc, uchar4 *im, int w, int h)`	Ядро	Вычисление принадлежности к определённому классу
`int main()`	Функция	Стартовая функция

Результаты

№ теста	sizesrc		Число блоков в сетке	Число потоков в блоке	Время, ms	
	x	y			GPU	CPU
1	5	5	1	32	7.20e-03	1.00e+00
2				256	3.01e-03	
3				512	3.97e-03	
4				1024	4.10e-03	
5			256	32	4.10e-03	
6				256	4.13e-03	
7				512	4.19e-03	
8				1024	4.10e-03	
9			512	32	4.16e-03	
10				256	4.16e-03	
11				512	4.16e-03	
12				1024	6.08e-03	
13			1024	32	4.13e-03	
14				256	4.16e-03	
15				512	5.73e-03	
16				1024	9.54e-03	
17	100	100	1	32	4.33e-01	2.43e+02
18				256	2.31e-01	
19				512	2.33e-01	
20				1024	2.32e-01	
21			256	32	1.06e-02	
22				256	1.45e-02	
23				512	2.64e-02	
24				1024	2.65e-02	
25			512	32	1.71e-02	
26				256	1.44e-02	

27			512	512	2.55e-02	
28				1024	2.65e-02	
29			1024	32	1.02e-02	
30				256	1.49e-02	
31				512	2.55e-02	
32				1024	2.65e-02	
33	10000	10000	1	32	3.33e+03	2.42e+06
34				256	1.70e+03	
35				512	1.70e+03	
36				1024	1.70e+03	
37			256	32	5.32e+01	
38				256	5.32e+01	
39				512	5.32e+01	
40				1024	5.32e+01	
41			512	32	5.42e+01	
42				256	5.37e+01	
43				512	5.37e+01	
44				1024	5.36e+01	
45			1024	32	5.36e+01	
46				256	5.19e+01	
47				512	5.19e+01	
48				1024	5.19e+01	

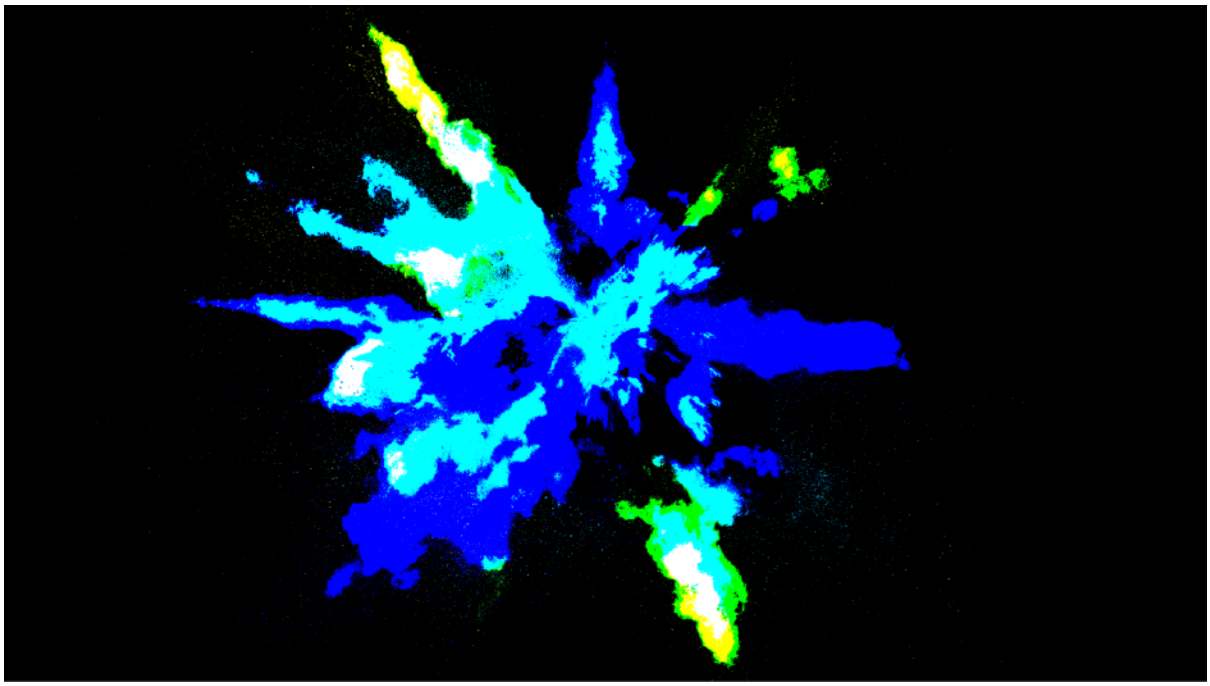
Примеры

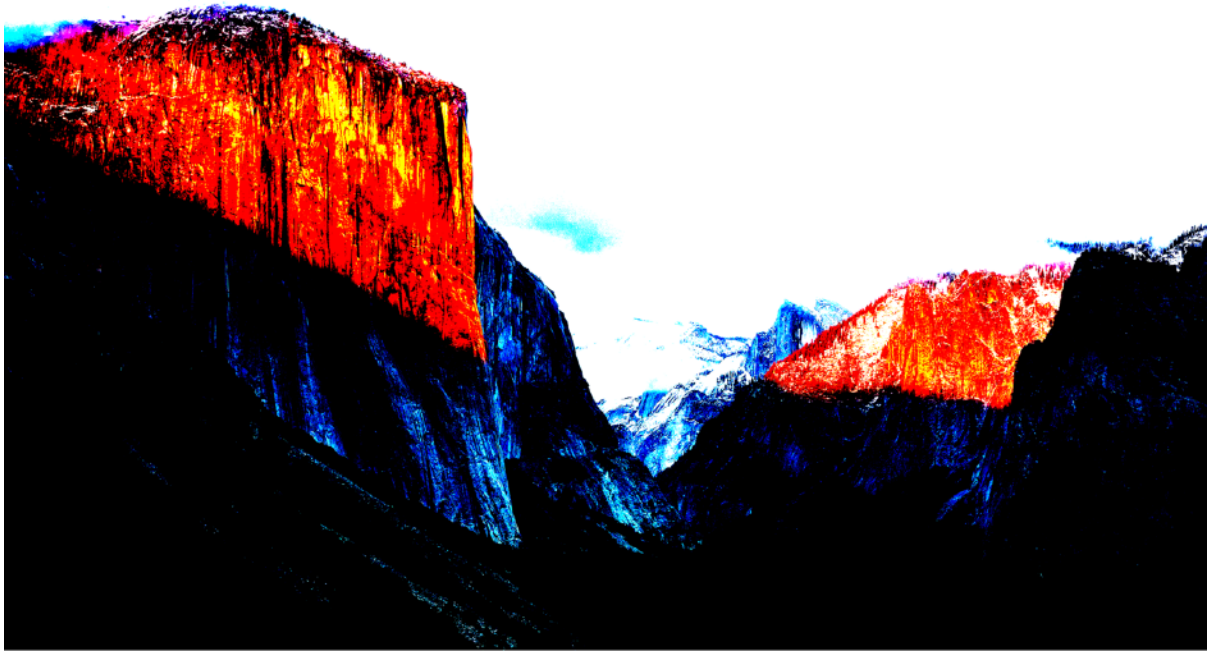
Для графических примеров использовался следующий массив средних и оценок ковариационной матрицы:

```
double3 avg[] = {   0,   0,   0,      // чёрный
                    0,   0, 255,      // синий
                    0, 255,   0,      // зелёный
                    0, 255, 255,      // цвет морской волны (бирюзовый)
                    255,   0,   0,    // красный
                    255,   0, 255,    // пурпурный (magenta)
                    255, 255,   0,    // жёлтый
                    255, 255, 255 };  // белый

double avg[] = { 1, 0, 0, 0, 1, 0, 0, 0, 1,
```

```
1, 0, 0, 0, 1, 0, 0, 0, 1,
1, 0, 0, 0, 1, 0, 0, 0, 1,
1, 0, 0, 0, 1, 0, 0, 0, 1,
1, 0, 0, 0, 1, 0, 0, 0, 1,
1, 0, 0, 0, 1, 0, 0, 0, 1,
1, 0, 0, 0, 1, 0, 0, 0, 1,
1, 0, 0, 0, 1, 0, 0, 0, 1 };
```





Выводы

Алгоритм может пригодится при решении многоклассовой задачи классификации, не обязательно пикселей изображений. Применительно к пикселям, алгоритм может помочь при решении задачи распознавания снимков со спутника.

В ходе лабораторной выяснилось, что на *GPU* лучше всего выполнять однотипные операции для большой размерности матриц. Т.к. они выполняются параллельно, то и суммарное время работы алгоритма заметно снижается.