

**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: Полей-Добронравова
Амелия

Группа: 8О-407Б

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 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.

В вариантах 1-4, формат входных данных одинаковый. 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. $nc \leq 32$, $np_j \leq 2^{19}$, $w*h \leq 4 * 10^8$.

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.

Вариант 5. Метод k-средних.

Входные данные. На первой строке задается путь к исходному изображению, на второй, путь к конечному изображению. На следующей строке, число nc -- кол-во кластеров. Далее идут nc строчек описывающих начальные центры кластеров. Каждая i -ая строчка содержит пару чисел -- координаты пикселя который является центром. $nc \leq 32$.

Пример:

Входной файл	hex: in.data	hex: out.data
in.data	03000000 03000000	03000000 03000000
out.data	A2DF4C00 F7C9FE00 9ED84500	A2DF4C01 F7C9FE00 9ED84501
2	B4E85300 99D14D00 92DD5600	B4E85301 99D14D01 92DD5601
2 2	A9E04C00 F7D1FA00 D4D0E900	A9E04C01 F7D1FA00 D4D0E900
1 1		

Входной файл	hex: out.data
--------------	---------------

in.data	08000000	08000000		
out.data	D2E27502	CFF65201	D3ED5701	D6E76902
5	C8F35B01	8E168200	CFF45001	AE977604
5 0	D3DC7102	7D1E7B00	AB9A8004	D9E58602
2 0	AB967E04	AE9D8004	87058200	D0F95B01
2 4	74148000	D0F55901	86136C00	85077400
6 2	D6E27702	D3609F03	D1609F03	CC5EA103
5 1	CC739D03	7C127F00	AA988804	AFA07D04
	D0E37702	7D117A00	D6EB5901	D6E37C02
	C9F85701	D655A103	D7EA7402	93127D00
	D35BA403	D4DD7902	B0A18404	D6DE7502
	D765A903	AD928404	D0D87C02	D7E97F02
	CD509E03	CAF85201	CFF75601	CEF45E01
	D0E86902	D1D17F02	AD928104	AFA18304
	D4DB5C01	88077D00	C6F75701	7D127D00
	A99A8E04	C8609E03	D15DA503	AB957E04
	AE9A8004	79218100	D065A103	A99E9A04

Программное и аппаратное обеспечение

Компилятор nvcc версии 7.0(g++ версии 4.8.4) на 64-х битной Ubuntu 14.04 LTS.

Параметры графического процессора:

Compute capability : 6.1

Name : GeForce GTX 1050

Total Global Memory : 2096103424

Shared memory per block : 49152

Registers per block : 65536

Max threads per block : (1024, 1024, 64)

Max block : (2147483647, 65535, 65535)

Total constant memory : 65536

Multiprocessors count : 5

Метод решения

__constant__ pair u[32]; константная память для центроидов кластеров, 32 - максимальное значение центроидов. Используется для обработки на GPU.

pair centre[n]; - массив для CPU до пересчета пикселей.

pair centre_new[n]; - массив для CPU после пересчета пикселей.

Программа заканчивается, когда эти два массива равны.

Пересчет кластеров для пикселей идёт с помощью выбора того центроида, до которого расстояние (по RGB) от пикселя наименьшее.

Пересчет центроида идёт через среднее арифметическое (по RGB) точек, принадлежащих этому кластеру.

Описание программы

Макрос **CSC** - макрос для отслеживания ошибок со стороны GPU, вызывается около функций для cuda и выводит текст ошибки при cudaError_t не равным cudaSuccess.

kernel - пересчет точек для кластеров, на GPU.

main - ввод данных, подготовка для передачи данных kernel для обработки, смещение центроидов кластеров, вывод результата.

dist - евклидово расстояние, функция только для GPU. Важно, что она работает только с входными переменными с типами данных double, потому что sqrt с входными переменными с типами данных int только для CPU.



pair - структура данных, по сути double3.

__constant__ pair u[32]; константная память для центроидов кластеров, 32 - максимальное значение центроидов.

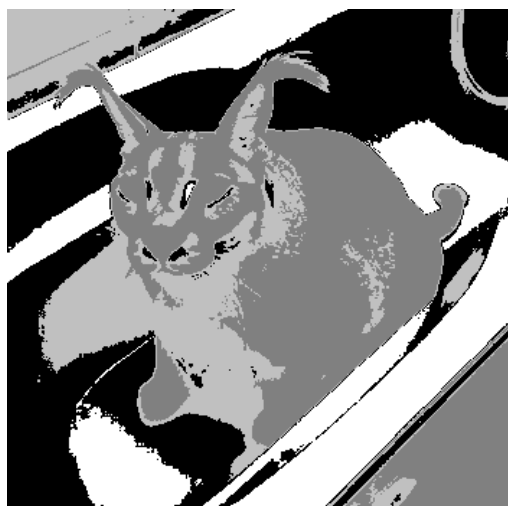
pair centre[n]; - массив для CPU до пересчета пикселей.

pair centre_new[n]; - массив для CPU после пересчета пикселей.

Результаты

Тест:	Результат на GPU:	на CPU:
 <p>in.data out.data 3 200 10 200 150 200 300</p>	 <p>kernel = «<1, 32>», time = 63.393215 kernel = «<1, 64>», time = 0.002432 kernel = «<1, 128>», time = 0.002400 kernel = «<1, 256>», time = 0.002432 kernel = «<1, 512>», time = 0.002432 kernel = «<1, 1024>», time = 0.002400 kernel = «<2, 32>», time = 0.002432 kernel = «<2, 64>», time = 0.002432 kernel = «<2, 128>», time = 0.002432 kernel = «<2, 256>», time = 0.002400 kernel = «<2, 512>», time = 0.002432 kernel = «<2, 1024>», time = 0.002400 kernel = «<4, 32>», time = 0.002400 kernel = «<4, 64>», time = 0.002432 kernel = «<4, 128>», time = 0.002400 kernel = «<4, 256>», time = 0.002432 kernel = «<4, 512>», time = 0.002400 kernel = «<4, 1024>», time = 0.002400 kernel = «<8, 32>», time = 0.002400 kernel = «<8, 64>», time = 0.002432</p>	<p>65</p>

	kernel = «<8, 128»», time = 0.002432 kernel = «<8, 256»», time = 0.002400 kernel = «<8, 512»», time = 0.002432 kernel = «<8, 1024»», time = 0.002400 kernel = «<16, 32»», time = 0.002432 kernel = «<16, 64»», time = 0.002400 kernel = «<16, 128»», time = 0.002432 kernel = «<16, 256»», time = 0.002400 kernel = «<16, 512»», time = 0.002432 kernel = «<16, 1024»», time = 0.002432 kernel = «<32, 32»», time = 0.002432 kernel = «<32, 64»», time = 0.002432 kernel = «<32, 128»», time = 0.002400 kernel = «<32, 256»», time = 0.002432 kernel = «<32, 512»», time = 0.002432 kernel = «<32, 1024»», time = 0.002400 kernel = «<64, 32»», time = 0.002432 kernel = «<64, 64»», time = 0.002400 kernel = «<64, 128»», time = 0.002400 kernel = «<64, 256»», time = 0.002432 kernel = «<64, 512»», time = 0.002432 kernel = «<64, 1024»», time = 0.002432 kernel = «<128, 32»», time = 0.002432 kernel = «<128, 64»», time = 0.002400 kernel = «<128, 128»», time = 0.002400 kernel = «<128, 256»», time = 0.002400 kernel = «<128, 512»», time = 0.002432 kernel = «<128, 1024»», time = 0.002400 kernel = «<256, 32»», time = 0.002432 kernel = «<256, 64»», time = 0.002400 kernel = «<256, 128»», time = 0.002432 kernel = «<256, 256»», time = 0.002432 kernel = «<256, 512»», time = 0.002496 kernel = «<256, 1024»», time = 0.002400 kernel = «<512, 32»», time = 0.002432 kernel = «<512, 64»», time = 0.002432 kernel = «<512, 128»», time = 0.002432 kernel = «<512, 256»», time = 0.002400 kernel = «<512, 512»», time = 0.002432 kernel = «<512, 1024»», time = 0.002400 kernel = «<1024, 32»», time = 0.002432 kernel = «<1024, 64»», time = 0.002400 kernel = «<1024, 128»», time = 0.002432 kernel = «<1024, 256»», time = 0.002400 kernel = «<1024, 512»», time = 0.002432 kernel = «<1024, 1024»», time = 0.002400	
--	--	--



in.data
out.data
4
150 150
350 350
0 350
0 0

kernel = «<1, 32>», time = 213.011749
kernel = «<1, 64>», time = 0.002400
kernel = «<1, 128>», time = 0.002400
kernel = «<1, 256>», time = 0.002336
kernel = «<1, 512>», time = 0.002432
kernel = «<1, 1024>», time = 0.002592
kernel = «<2, 32>», time = 0.002400
kernel = «<2, 64>», time = 0.002432
kernel = «<2, 128>», time = 0.008384
kernel = «<2, 256>», time = 0.002528
kernel = «<2, 512>», time = 0.002432
kernel = «<2, 1024>», time = 0.002464
kernel = «<4, 32>», time = 0.002432
kernel = «<4, 64>», time = 0.002400
kernel = «<4, 128>», time = 0.002368
kernel = «<4, 256>», time = 0.002400
kernel = «<4, 512>», time = 0.002400
kernel = «<4, 1024>», time = 0.002400
kernel = «<8, 32>», time = 0.002400
kernel = «<8, 64>», time = 0.002496
kernel = «<8, 128>», time = 0.002432
kernel = «<8, 256>», time = 0.002400
kernel = «<8, 512>», time = 0.002464
kernel = «<8, 1024>», time = 0.002432
kernel = «<16, 32>», time = 0.002400
kernel = «<16, 64>», time = 0.002432
kernel = «<16, 128>», time = 0.002400
kernel = «<16, 256>», time = 0.002400
kernel = «<16, 512>», time = 0.002400
kernel = «<16, 1024>», time = 0.002432
kernel = «<32, 32>», time = 0.002400
kernel = «<32, 64>», time = 0.002400
kernel = «<32, 128>», time = 0.002432
kernel = «<32, 256>», time = 0.002432
kernel = «<32, 512>», time = 0.002432
kernel = «<32, 1024>», time = 0.002400
kernel = «<64, 32>», time = 0.002400
kernel = «<64, 64>», time = 0.002400
kernel = «<64, 128>», time = 0.007680
kernel = «<64, 256>», time = 0.002400
kernel = «<64, 512>», time = 0.002432

	kernel = «<64, 1024»», time = 0.002400 kernel = «<128, 32»», time = 0.002592 kernel = «<128, 64»», time = 0.002432 kernel = «<128, 128»», time = 0.002432 kernel = «<128, 256»», time = 0.002400 kernel = «<128, 512»», time = 0.002432 kernel = «<128, 1024»», time = 0.002240 kernel = «<256, 32»», time = 0.007712 kernel = «<256, 64»», time = 0.002432 kernel = «<256, 128»», time = 0.002400 kernel = «<256, 256»», time = 0.002432 kernel = «<256, 512»», time = 0.002432 kernel = «<256, 1024»», time = 0.002432 kernel = «<512, 32»», time = 0.002464 kernel = «<512, 64»», time = 0.002400 kernel = «<512, 128»», time = 0.002432 kernel = «<512, 256»», time = 0.002400 kernel = «<512, 512»», time = 0.002400 kernel = «<512, 1024»», time = 0.002400 kernel = «<1024, 32»», time = 0.002400 kernel = «<1024, 64»», time = 0.002400 kernel = «<1024, 128»», time = 0.002464 kernel = «<1024, 256»», time = 0.002432 kernel = «<1024, 512»», time = 0.002464 kernel = «<1024, 1024»», time = 0.002432	
--	--	--

Код программы для CPU:

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <vector>
#include <string>
#include <cmath>

typedef struct pair {
    double x;
    double y;
    double z;
} pair;

double dist(double x1, double y1, double z1, double x2, double y2, double z2) {
    return sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2) + (z1-z2)*(z1-z2));
}

pair centre[32];

void kernel(char** data, int w, int h, int n) {
    int x, y;
    double mas[32];
    for (y = 0; y < h; y++) {
        for (x = 0; x < w; x++) {
            double p[4];
            p[0] = double(data[y * w + x][0]);
            p[1] = double(data[y * w + x][1]);
            p[2] = double(data[y * w + x][2]);
            for (int j = 0; j < n; j++) {
```



```

        mas[j] = dist(centre[j].x, centre[j].y, centre[j].z, p[0], p[1], p[2]); //заполняем массив расстояний
    }
    //поиск минимума
    double MIN = mas[0];
    int min_i = 0;
    for (int j = 1; j < n; j++) {
        if (mas[j] < MIN) {
            MIN = mas[j];
            min_i = j;
        }
    }
    data[y*w+x][3] = min_i;
}
}
}

```

```

int main() {
    std::string inputFile;
    std::string outputFile;
    int n, w, h;
    int x, y;

    std::cin >> inputFile >> outputFile;
    scanf("%d", &n);
    pair centre_new[n];

    FILE* fp = fopen(inputFile.c_str(), "rb");

    fread(&w, sizeof(int), 1, fp);
    fread(&h, sizeof(int), 1, fp);
    char** data = (char**)malloc(sizeof(char*) * w * h);
    for (int i = 0; i < w*h; i++) {
        data[i] = (char*)malloc(sizeof(char) * 4);
    }
    for(int i = 0; i < w*h; i++) {
        fread(data[i], sizeof(char) * 4, w * h, fp);
    }
    fclose(fp);

    for(int i = 0; i < n; i++) {
        scanf("%d %d", &x, &y);
        double p[4];
        p[0] = double(data[y * w + x][0]);
        p[1] = double(data[y * w + x][1]);
        p[2] = double(data[y * w + x][2]);
        centre[i].x = p[0];
        centre[i].y = p[1];
        centre[i].z = p[2];
    }
}

```

```

bool flag = true;
while(flag) {
    kernel(data, w, h, n);
    //printf("\n");
    //обновление центра кластеров
    int k[n];
    double sum[n][3];
}

```



```

for(y = 0; y < n; y++) {
    k[y] = 0;
    sum[y][0] = 0;
    sum[y][1] = 0;
    sum[y][2] = 0;
}
double t[4];
for(y = 0; y < h; y++) {
    for(x = 0; x < w; x++) {
        t[0] = double(data[y * w + x][0]);
        t[1] = double(data[y * w + x][1]);
        t[2] = double(data[y * w + x][2]);
        t[3] = double(data[y * w + x][3]);
        //printf("%d ", t);
        k[int(t[3])] += 1;
        sum[int(t[3])][0] += t[0];
        sum[int(t[3])][1] += t[1];
        sum[int(t[3])][2] += t[2];
    }
}
//printf("\n");
for(int r = 0; r < n; r++) {
    if (k[r] > 0) {
        centre_new[r].x = sum[r][0] / k[r];
        centre_new[r].y = sum[r][1] / k[r];
        centre_new[r].z = sum[r][2] / k[r];
    }
}

//условие прекращения
for(int i = 0; i < n; i++) {
    if (!(centre[i].x == centre_new[i].x && centre[i].y == centre_new[i].y && centre[i].z == centre_new[i].z)) {
        flag = false; //были не одинаковые
        break;
    }
}
if (flag == false) {
    flag = true; //продолжаем обработку
    for(int i = 0; i < n; i++) {
        centre[i].x = centre_new[i].x;
        centre[i].y = centre_new[i].y;
        centre[i].z = centre_new[i].z;
    }
}
else {
    flag = false; //заканчиваем, выход из цикла
}
}

fp = fopen(outputFile.c_str(), "wb");
fwrite(&w, sizeof(int), 1, fp);
fwrite(&h, sizeof(int), 1, fp);
for(int i = 0; i < w*h; i++) {
    fwrite(data[i], sizeof(char) * 4, w * h, fp);
}
fclose(fp);

free(data);

```

```
return 0;  
}
```

Выводы

1. Константная память достаточно быстрая из доступных GPU. Есть возможность записи данных с хоста, но при этом в пределах GPU возможно только чтение из этой памяти. Для размещения данных в константной памяти предусмотрен спецификатор `__constant__`.
2. Если необходимо использовать массив в константной памяти, то его размер необходимо указать заранее, так как динамическое выделение в отличие от глобальной памяти в константной не поддерживается.
3. Для записи с хоста в константную память используется функция `cudaMemcpyToSymbol`, и для копирования с device на хост `cudaMemcpyFromSymbol`.
4. В представленных мной тестах время на разных ядрах почти не отличается, потому что алгоритм очень быстро заканчивает работу: выбранные мной начальные точки центроидов кластеров почти идеально выбраны по цветам, количество кластеров очень маленькое и палитра цветов пикселей картинок очень ограничена.