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: М.А. Бронников

Группа: 8О-407Б

Преподаватели: К.Г. Крашенинников,

А.Ю. Морозов

Москва, 2020

**Условие**

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

**Вариант 4.** Метод максимального правдоподобия.

**Программное и аппаратное обеспечение**

Device: GeForce GT 545

Размер глобальной памяти: 3150381056

Размер константной памяти : 65536

Размер разделяемой памяти: 49152

Регистров на блок: 32768

Максимум потоков на блок: 1024

Количество мультипроцессоров : 3

OS: Linux Mint 20 Cinnamon

Редактор: VSCode

**Метод решения**

Для начала необходимо посчитать значения средних по каждому из каналов и значения обратной матрицы ковариации для каждого из заданных признаков. Далее для каждого из пикселей изображения определяется его класс по формуле:



**Описание программы**

Для выполнения программы я реализовал собственный класс изображения в методе которого и вызывался kernel. Этот класс не потерпел значительных изменений со времени выполнения второй лабораторной работы.

Для выполнения операции я инициализировал на этапе компиляции массив константной памяти необходимого размера, а именно максимального количества возможных классов, умноженного на размер, необходимый для хранения вычислительной информации для каждого из классов.

struct class\_data{

float avg\_red;

float avg\_green;

float avg\_blue;

float cov11;

float cov12;

float cov13;

float cov21;

float cov22;

float cov23;

float cov31;

float cov32;

float cov33;

float log\_det;

};

// constant memory

\_\_constant\_\_ class\_data computation\_data[MAX\_CLASS\_NUMBERS];

Для копирования данных с host в этот участок памяти у меня есть следующий код:

throw\_on\_cuda\_error(

cudaMemcpyToSymbol(computation\_data, cov\_avg,

MAX\_CLASS\_NUMBERS\*sizeof(class\_data), 0, cudaMemcpyHostToDevice)

);

После чего Сами же данные для вычисления я расчитываю на CPU, поскольку расчет на GPU не даст значимого прироста к производительности.

После чего я вызываю kernel с заданным количеством блоков и потоков, где я преобразую изображение в кластеризованное по описанному алгоритму:

dim3 threads = dim3(MAX\_X, MAX\_Y);

dim3 blocks = dim3(BLOCKS\_X, BLOCKS\_Y);

classification<<<blocks, threads>>>(d\_data, \_height, \_widht, indexes.size());

throw\_on\_cuda\_error(cudaGetLastError());

В самом kernel мы вычисляем правдоподобие для пикселя по каждому из классов и записываем наиболее вероятный номер класса.

\_\_global\_\_ void classification(uint32\_t\* picture, uint32\_t h, uint32\_t w, uint8\_t classes){

uint32\_t idx = blockIdx.x \* blockDim.x + threadIdx.x;

uint32\_t idy = blockIdx.y \* blockDim.y + threadIdx.y;“

uint32\_t step\_x = blockDim.x \* gridDim.x;

uint32\_t step\_y = blockDim.y \* gridDim.y;

// run for axis y

for(uint32\_t i = idy; i < h; i += step\_y){

// run for axis x

for(uint32\_t j = idx; j < w; j += step\_x){

// init very big num

float min = INT32\_MAX;

uint32\_t pixel = picture[i\*w + j];

uint8\_t ans\_c = 0;

for(uint8\_t c = 0; c < classes; ++c){

float red = RED(pixel);

float green = GREEN(pixel);

float blue = BLUE(pixel);

float metric = 0.0;

red -= computation\_data[c].avg\_red;

green -= computation\_data[c].avg\_green;

blue -= computation\_data[c].avg\_blue;

float temp\_red = red\*computation\_data[c].cov11 +

green\*computation\_data[c].cov21 + blue\*computation\_data[c].cov31;

float temp\_green = red\*computation\_data[c].cov12 +

green\*computation\_data[c].cov22 + blue\*computation\_data[c].cov32;

float temp\_blue = red\*computation\_data[c].cov13 +

green\*computation\_data[c].cov23 + blue\*computation\_data[c].cov33;

// dot + log(|cov|)

metric = (temp\_red\*red + temp\_green\*green + temp\_blue\*blue + computation\_data[c].log\_det);

if(metric < min){

ans\_c = c;

min = metric;

}

}

#ifndef \_\_WITH\_IMG\_\_

pixel ^= ((uint32\_t) ans\_c) << 24;

#endif

picture[i\*w + j] = pixel;

}

}

}

После вызова kernel я копирую данные в массив и освобождаю выделенную память.

**Результаты**

Для иллюстрации результатов работы алгоритма я выбрал 2 изображения:





После чего я применил конвертер в заданный в задании формат, который я взял из материалов от преподавателя. После чего я применил к ним свою программу, только вместо номера класса на место alpha канала я записываю avg по наиболее вероятному классу. Полученные результаты:



****

В обоих изображениях я выделял несколько точек из трех классов. На лицо некоторые неточности, которые устраняются с добавлением новых классов и новых точек для них.

**Таблица замеров времени выполнения**

|  |  |  |
| --- | --- | --- |
| CPU или конфигурация | Размер изображения | Время выполнения |
| CPU | 800 1200 | 99.141 |
| CPU | 1600 2560 | 290.946 |
| <<<(4, 4), (4, 4)>>> | 800 1200 | 7.38314 |
| <<<(4, 4), (4, 4)>>> | 1600 2560 | 29.7029 |
| <<<(4, 4), (16, 16)>>> | 800 1200 | 1.37098 |
| <<<(4, 4), (16, 16)>>> | 1600 2560 | 5.76128 |
| <<<(4, 4), (32, 32)>>> | 800 1200 | 1.22864 |
| <<<(4, 4), (32, 32)>>> | 1600 2560 | 5.04371 |
| <<<(16, 16), (4, 4)>>> | 800 1200 | 5.1625 |
| <<<(16, 16), (4, 4)>>> | 1600 2560 | 21.9541 |
| <<<(16, 16), (16, 16)>>> | 800 1200 | 2.1607 |
| <<<(16, 16), (16, 16)>>> | 1600 2560 | 6.9639 |
| <<<(16, 16), (32, 32)>>> | 800 1200 | 1.24474 |
| <<<(16, 16), (32, 32)>>> | 1600 2560 | 4.74365 |
| <<<(32, 32), (4, 4)>>> | 800 1200 | 5.01814 |
| <<<(32, 32), (4, 4)>>> | 1600 2560 | 21.536 |
| <<<(32, 32), (16, 16)>>> | 800 1200 | 1.11478 |
| <<<(32, 32), (16, 16)>>> | 1600 2560 | 4.46362 |
| <<<(32, 32), (32, 32)>>> | 800 1200 | 1.50403 |
| <<<(32, 32), (32, 32)>>> | 1600 2560 | 5.12384 |

**Выводы**

В ходе лабораторной работы я познакомился с одним из методов сегментации изображения. Такие методы сейчас набирают особенную популярность с развитием нейронных сетей и компьютерного зрения в частности.

Реализованный мной алгоритм, несмотря на то, что неплохо классифицирует пиксели на изображении, имеет некоторые недостатки, главный из которых — необходимость вручную задавать классы и некоторое количество пикселей, которое к нему относиться. Так как зачастую требуется просто выделить некоторые кластеры, вне зависимости от начального разбиения, имеет смысл обратиться к одному из методов «без учителя», такому как k-means.

В ходе выполнения работы возникла трудность в том, что я более суток не мог понять, что логарифм в формуле функции правдоподобия берется не от нормы матрицы, а модуля ее определителя. Этот момент был для меня не очевидным и потратил много моих сил и нервов.