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: Е.С. Пищик
Группа: 8О-406Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2022

Условие

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

Вариант 1. Метод максимального правдоподобия.

Программное и аппаратное обеспечение

GPU via SSH:

- Compute capability: 2.1
- Name: GeForce GT 545
- Total Global Memory: 3150381056
- Shared memory per block: 49152
- Registers per block: 32768
- Warp size: 32
- Max threads per block: (1024, 1024, 64)
- Max block: (65535, 65535, 65535)
- Total constant memory: 65536
- Multiprocessors count: 3

CPU via SSH: Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
RAM via SSH: 16 Gb.
HDD via SSH: 471 Gb.
OS: Windows 10
OS via SSH: Ubuntu 16.04.6 LTS
IDE: Visual Studio Code
Compiler via SSH: nvcc

Метод решения

На этапе предобработки вычисляем для каждого класса среднее, обратную матрицу ковариации, логарифм от модуля определителя матрицы ковариации.
Проходим по каждому пикселю изображения, для каждого пикселя вычисляем новое значение яркости по заданной формуле при помощи значений r,g,b данного пикселя и значений полученных на этапе предобработки.

Описание программы

Программа состоит из одного основного файла gpu.cu. Файл состоит из функций inverse, preprocessing, kernel, main. inverse - функция находит обратную матрицу ковариации. preprocessing - вычисляются все значения, которые могут быть найдены перед запуском ядра. main - основная функция программы, здесь происходит считывание входных данных, выделение памяти под изображение на host устройстве, выделение памяти под изображение на device устройстве (при помощи функций cudaMalloc), копирование данных с host на device (при помощи cudaMemcpy). При помощи функций cudaEventCreate, cudaEventRecord, cudaEventSynchronize происходит измерение времени

работы алгоритма. После всех операций по выделению памяти, копированию векторов, созданию переменных для подсчета времени запускается функция kernel на device устройстве. После выполнения алгоритма идет вывод в потом ошибок времени затраченного на алгоритм, очищение памяти на host устройстве и device устройстве (при помощи cudaFree). В самом начале мы создаем три массива в константной памяти, где будем хранить предпосчитанные значения.

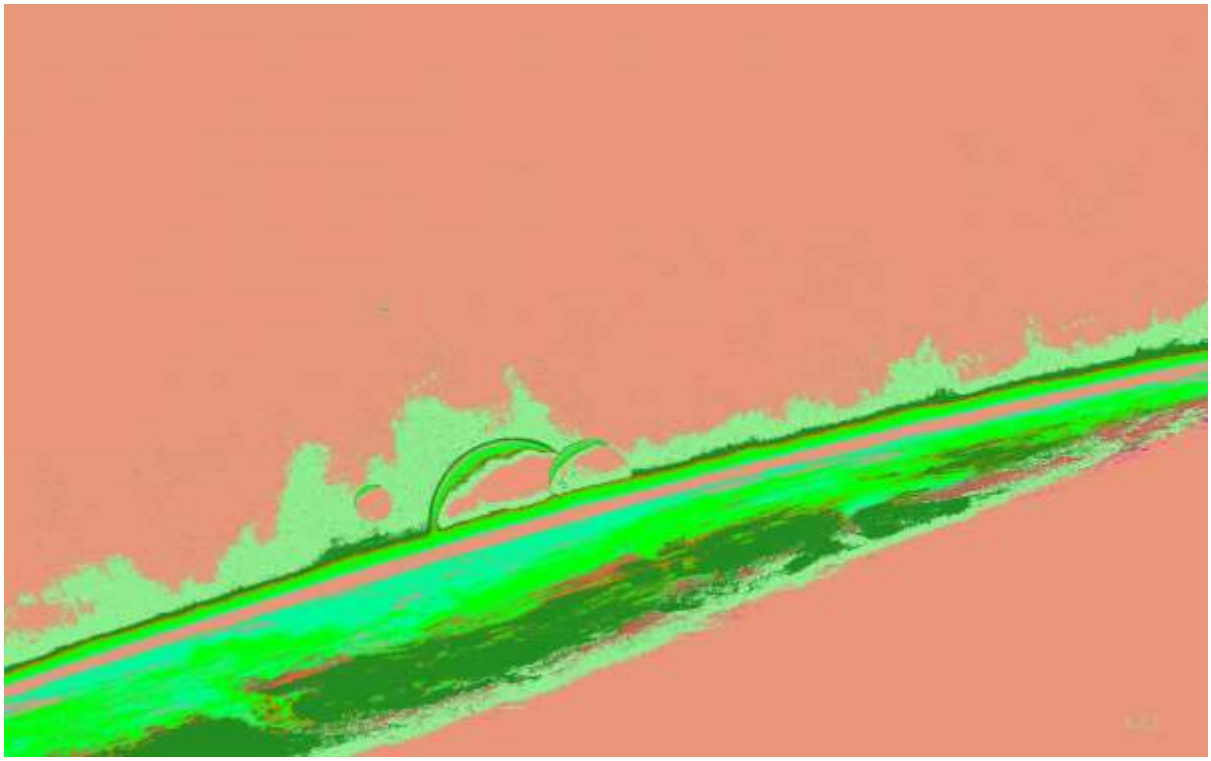
В функции ядра, мы просто вычисляем абсолютный номер потока в одномерной сетке - переменная idx. А также количество всех наших - переменная offsetx. В самом ядре мы просто проходим по пикселям при помощи idx и вычисляем по формуле новое значение яркости для данного пикселя.

```
__global__ void kernel(uchar4 *out, int w, int h, int nc)
{
   int idx = blockDim.x * blockIdx.x + threadIdx.x;
   int offsetx = blockDim.x * gridDim.x;

   uchar4 p;
        int argmax;
        double maxv, v;
        vec3 curr_sub0, curr_sub1;

        for(int x = idx; x < w * h; x += offsetx) {
                argmax = -1;
                maxv = -DBL_MAX;

                for(int cls = 0; cls < nc; ++cls) {
                        p = out[x];
                        curr_sub0.r = p.x - d_avg[cls].r;
                        curr_sub0.g = p.y - d_avg[cls].g;
                        curr_sub0.b = p.z - d_avg[cls].b;
                        curr_sub1.r = curr_sub0.r * d_cov[cls].arr[0][0] + curr_sub0.g *
d_cov[cls].arr[1][0] + curr_sub0.b * d_cov[cls].arr[2][0];
                        curr_sub1.g = curr_sub0.r * d_cov[cls].arr[0][1] + curr_sub0.g *
d_cov[cls].arr[1][1] + curr_sub0.b * d_cov[cls].arr[2][1];
                        curr_sub1.b = curr_sub0.r * d_cov[cls].arr[0][2] + curr_sub0.g *
d_cov[cls].arr[1][2] + curr_sub0.b * d_cov[cls].arr[2][2];
                        v = -(curr_sub0.r * curr_sub1.r + curr_sub0.g * curr_sub1.g +
curr_sub0.b * curr_sub1.b) - d_logv[cls];
                        if(v > maxv) {
                                maxv = v;
                                argmax = cls;
                        }
                }
```

```
                out[x].w = argmax;
        }
}
```

Результаты

Результат классификации изображения 5000*8000 с количеством классов с количеством классов 32, количество пикселей для каждого класса npj генерировалось рандомно из диапазона [10000, 524288], значения x и y каждого из пикселей для описания классов генерировались рандомно из диапазонов [0, 8000] и [0, 5000].



В таблице приведены результаты сравнения для данного теста на разных конфигурациях ядер и на cpu.

CPU	19.628 s
GPU<<<1,32>>>	17.692 s
GPU<<<8,32>>>	2.639 s
GPU<<<32,32>>>	1.639 s
GPU<<<64,64>>>	1.380 s
GPU<<<128,128>>>	1.331 s
GPU<<<256,256>>>	1.331 s
GPU<<<512,512>>>	1.350 s

Выводы

В данной лабораторной работе я изучил различные виды памяти на GPU, в частности научился работать с константной памятью. Использовал одномерную сетку потоков для обработки двумерных данных, мы можем так сделать, потому что не используем информацию о соседних пикселях для вычисления яркости каждого пикселя, изучил алгоритм максимального правдоподобия.