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.

Выполнила: К.О. Михеева
Группа: 8О-407Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2023

Условие

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

Вариант задания: Вариант 4. Метод спектрального угла.

Программное и аппаратное обеспечение

Name:	Tesla T4
Compute capability:	7.5
Dedicated video memory:	15835 MB
Shared memory per block:	49152 bytes
Register per block:	65536 bytes
Total constant memory:	65536
Max threads per multiprocessor:	2048
Max threads per block:	1024
Multiprocessors const:	40

AMD RYZEN 5 5500U

Architecture:	Ryzen 5 Zen2(Lucienne)
Processor Technology for CPU Cores:	TSMC 7nm FinFET
CPU Cores:	6
Thread(s):	12
CPU Socket	FP6
CPU MHz:	3600
SMP # CPUs:	1
L2 cache: L3 cache:	3MB 8MB
RAM	16GB
SSD	512GB

OS: Windows 10 - 64-Bit Edition, Ubuntu x86 64-Bit;
IDE: VS code;
Compiler: nvcc

Метод решения

Программа загружает входное изображение, определяет количество кластеров (nc), создает и обновляет средние цвета (RGB) для каждого кластера, нормализует их, вычисляет схожесть пикселей с кластерами на основе нормализованных средних цветов, присваивает каждому пикселю метку кластера в соответствии с максимальной схожестью, и сохраняет результат в выходном изображении.

Источником для выполнения данной работы является пример данный на лекции преподавателем.

Описание программы

Данная программа состоит из одного главного файла. В функции main происходит считывание исходных данных для дальнейшей обработки, а после завершения работы с ней выводит новые обработанные данные.

Основными типами данных являются RGBColor (структура для представления цвета в формате RGB с компонентами x, y и z), Point2D (структура для представления двумерной точки с координатами x и y), uchar4 (тип данных для представления цветовых компонент пикселей изображения).

Для начала заведем константные массивы средних значений и нормирования для того, чтобы обеспечить доступность данных на всех блоках и нитях без необходимости повторной передачи на каждом вызове ядра. Пропишем основные функции.

Первая из них является CalculateAverages, которая вычисляет средний цвет для каждого кластера на основе пикселей, принадлежащих этому кластеру в изображении. Она использует вектор avgs для накопления суммы цветов пикселей в каждом кластере и затем делит эту сумму на количество пикселей, чтобы получить средний цвет. Полученные средние цвета сохраняются в массиве cAvgs для последующего использования.

Вторая эта функция нормализации NormalizeAverages. Происходит цикл по количеству кластеров (nc). Для каждого кластера выполняются следующие действия: сначала получается средний цвет кластера (avg) из массива cAvgs, вычисляется магнитуда (длина) вектора среднего цвета с использованием формулы: magnitude = sqrt(avg.x * avg.x + avg.y * avg.y + avg.z * avg.z), затем средний цвет нормализуется путем деления каждой компоненты (x, y, и z) на его магнитуду, и результат сохраняется в массиве cNormAvgs.

Далее Функция Similarity принимает пиксель ps и идентификатор кластера classId, оценивает схожесть пикселя с указанным кластером, вычисляя скалярное произведение между нормализованными цветами пикселя и кластера, что позволяет определить степень схожести пикселя с данным кластером.

Ядро SpectralClustering выполняет спектральную кластеризацию изображения с использованием CUDA. Для каждого пикселя оценивается схожесть с каждым из кластеров, и пиксель присваивается кластеру с наибольшей схожестью. Результат

записывается в компоненту w пикселя out, что позволяет разделить изображение на кластеры на основе схожести цветовых характеристик.

Код функции

```
struct RGBColor {
   double x, y, z;
};

__device__ __host__ void RGB(RGBColor& rgb, uchar4* ps) {
   rgb = {static_cast<double>(ps->x), static_cast<double>(ps->y),
static_cast<double>(ps->z)};
}

struct Point2D {
   int x, y;
};


__constant__ RGBColor constAvgs[32];
RGBColor cAvgs[32];


void CalculateAverages(vector<vector<Point2D>>&classPoints,
uchar4* image, int w, int h, int nc) {
   vector<RGBColor> avgs(32, {0, 0, 0});

   for (int i = 0; i < nc; i++) {
       int np = classPoints[i].size();
       for (int j = 0; j < np; j++) {
           Point2D point = classPoints[i][j];
           uchar4 ps = image[point.y * w + point.x];
           RGBColor rgb;
           RGB(rgb, &ps);

           avgs[i].x += rgb.x;
           avgs[i].y += rgb.y;
           avgs[i].z += rgb.z;
       }

       avgs[i].x /= np;
       avgs[i].y /= np;
       avgs[i].z /= np;
   }

   for (int i = 0; i < nc; i++) {
       cAvgs[i] = avgs[i];
   }
}
```

```
__constant__ RGBColor constNormAvgs[32];
RGBColor cNormAvgs[32];


void NormalizeAverages(int nc) {
   for (int i = 0; i < nc; i++) {
       RGBColor avg = cAvgs[i];
       double magnitude = sqrt(avg.x * avg.x + avg.y * avg.y +
avg.z * avg.z);
       cNormAvgs[i].x = avg.x / magnitude;
       cNormAvgs[i].y = avg.y / magnitude;
       cNormAvgs[i].z = avg.z / magnitude;
   }
}


__device__ double Similarity(uchar4 ps, int classId) {
   RGBColor rgb;
   RGB(rgb, &ps);

   RGBColor normAvg = constNormAvgs[classId];
   return rgb.x * normAvg.x + rgb.y * normAvg.y + rgb.z *
normAvg.z;
}


__global__ void SpectralClustering(uchar4* out, int w, int h, int
nc) {
   int idx = blockDim.x * blockIdx.x + threadIdx.x;
   int idy = blockDim.y * blockIdx.y + threadIdx.y;
   int offsetx = blockDim.x * gridDim.x;
   int offsety = blockDim.y * gridDim.y;
   int x, y;
    for (y = idy; y < h; y += offsety) {
       for (x = idx; x < w; x += offsetx) {
           double similarity = Similarity(out[y * w + x], 0);
           int cluster = 0;
           for (int i = 0; i < nc; i++) {
               double tsimilarity = Similarity(out[y * w + x], i);

               if (tsimilarity > similarity) {
                   similarity = tsimilarity;
                   cluster = i;
               }

           }
           out[y * w + x].w = (unsigned char)cluster;
       }
```

}

Результаты

Время исполнения программы в мс при различных данных
Размеры изображений:
В малом изображении -128 x 128;
В среднем изображении - 1600 x 1600;
В большом изображении - 7680 x 4320.

	Малое изображение	Среднее изображение	Большое изображение
<<<1, 32>>>, <<<132>>>	0.12	0.88	25.79
<<<32, 32>>>, <<<32, 32>>>	0.17	0.34	1.86
<<<1, 128>>>, <<<1, 128>>>	0.39	1.65	10.11
<<<128, 128>>>, <<<32, 32>>>	0.31	0.18	1.98
<<<1, 256>>>, <<<1, 256>>>	0.95	5.29	31.54
<<<256, 256>>>, <<<32,32>>>	0.34	1.07	3.01
<<<1, 512>>>, <<<1, 512>>>	1.74	6.18	6.95
<<<512, 512>>>, <<<32, 32>>>	0.17	3.05	7.94
<<<1, 1024>>>, <<<1, 1024>>>	0.28	5.87	41.04
CPU	0.41	32.79	589.05

1) До:

После:

2) До:

После:

3) До:

После:

Выводы

В данной лабораторной работе был представлен метод спектрального угла..

Метод основном применяется для обработке изображений и компьютерном зрении.

Задачи которые можно решить методом спектрального угла являются:

- Поиск похожих изображений, это полезно для поиска и классификации изображений в больших коллекциях фотографий, например, в поисковых системах.
- Обработка видеоданных, применение алгоритма кадр за кадром может использоваться для анализа движения и обнаружения объектов на видео.

В данной работе у меня возникли проблемы из-за непонимания данного алгоритма и его правильной реализации. Было сложно понять как правильно реализовать средние значение и номер класса по данным формулам из условия задачи. Также я забыла ввести второй не константный массив, чтобы можно было перезаписывать получившиеся данные.