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: Лобанов О. А.
Группа: 8О-408Б-20
Преподаватель: А.Ю. Морозов

Москва, 2023

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

Вариант 4. Метод спектрального угла.

Программное и аппаратное обеспечение

Device: Tesla T4
Compute capability: 7.5
Total constant memory: 65536
Registers per block: 65536
Max threads per block: 1024
Multiprocessors count: 40
OS: Ubuntu 22.04.2 LTS
Redactor: colab google

Метод решения

Оценка вектора средних:

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

Для некоторого пикселя *p*, номера класса *jc* определяется следующим образом:

$$jc = arg\,max_j \left[p^T * \frac{avg_j}{|avg_j|} \right]$$

Описание программы

Изначально я определяю средние значения и их нормирование. Это действие выполняется на CPU.
Следующим шагом я создал массив константной памяти, где будут находится результаты каждых классов.
В kernel происходит классификация для каждого пикселя по максимальному соответствию.
После того, как как для каждого пикселя был определен класс, записываю данные в файл.
Функция нахождения пикселя:

```
__device__ double find_pixel(uchar4 check, int number) {
    double result = 0;
    vector_2 kek;
    Get(&check, kek);

    double new_RGB[3];
    double new_norm[3];
```

```
    new_RGB[0] = kek.x;
    new_RGB[1] = kek.y;
    new_RGB[2] = kek.z;

    new_norm[0] = normis_avg[number].x;
    new_norm[1] = normis_avg[number].y;
    new_norm[2] = normis_avg[number].z;
    for (int i = 0; i < 3; i++) {
        result += new_RGB[i] * new_norm[i];
    }
    return result;
}
```

kernel:

```
__global__ void kernel(uchar4* data, int w, int h, int nc) {
    int xdx = blockDim.x * blockIdx.x + threadIdx.x;
    int ydy = blockDim.y * blockIdx.y + threadIdx.y;
    int offsetX = blockDim.x * gridDim.x;
    int offsetY = blockDim.y * gridDim.y;
    uchar4 check;

    for (int i = ydy; i < h; i += offsetY) {
        for (int j = xdx; j < w; j += offsetX) {
            check = data[i * w + j];
            double finded_1 = find_pixel(check, 0);
            int n = 0;
            for (int a = 1; a < nc; a++) {
                double finded_2 = find_pixel(check, a);
                if (finded_1 < finded_2) {
                    finded_1 = finded_2;
                    n = a;
                }
            }
            data[i * w + j].w = n;
        }
    }
}
```

Результаты
Пример работы алгоритма:

Картинка №1

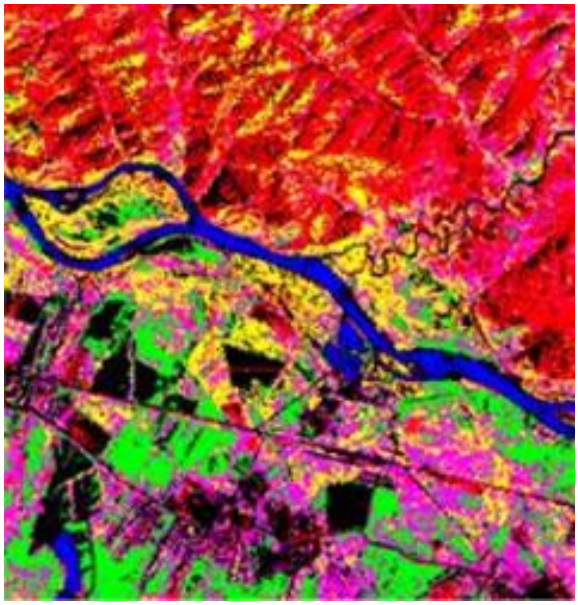

Таблица замеров времени

Конфигурация	Размер	Время
CPU	290 310	25.241
dim3(16, 16)	290 310	0.452
dim3(32, 32)	290 310	0.291
dim3(64, 64)	290 310	0.001
dim3(128, 128)	290 310	0.001
CPU	1200 2350	263.746
dim3(16, 16)	1200 2350	26.032
dim3(32, 32)	1200 2350	1.337
dim3(64, 64)	1200 2350	0.269
dim3(128, 128)	1200 2350	0.001

Выводы

В ходе выполнения работы я познакомился с одним из методов сегментации изображения. Реализовав мой алгоритм, я научился работать с константной памятью. В нее можно записать что – либо, используя функцию cudaMemcpyToSymbol.