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: *А. Ю. Голов*
Группа: *М8О-401*
Преподаватель: А.Ю. Морозов

Москва, 2025

Условие

Цель работы. Научиться использовать GPU для классификации и кластеризации изображений. Использование константной памяти и одномерной сетки потоков.
Вариант 3. Метод минимального расстояния.

Программное и аппаратное обеспечение

Дать характеристики графического процессора (compute capability, графическая память, разделяемая память, константная память, количество регистров на блок, максимальное количество блоков и нитей, количество мультипроцессоров), процессора, оперативной памяти и жесткого диска. Описать программное обеспечение (OS, IDE, compiler и тд.).
Видеокарта: NVIDIA GeForce RTX 2060 Mobile

- Compute Capability: 7.5
- Графическая память: 6 ГБ GDDR6, с 192-битной шиной и пропускной способностью 336 ГБ/с
- Разделяемая память: до 64 КБ на мультипроцессор
- Константная память: 64 КБ
- Количество регистров на блок: 65 536
- Максимальное количество блоков на мультипроцессор: 16
- Максимальное количество нитей на мультипроцессор: 1 024
- Количество мультипроцессоров (SM): 30

Процессор: Intel Core i7-9750H имеет следующие характеристики:

- Количество ядер: 6
- Количество потоков: 12
- Техпроцесс: 14 нм

Оперативная память:

- Объём: 16 ГБ
- Тактовая частота: 3500 MHz
- Поколение: DDR4

Жёсткий диск:

- Объём: 512 ГБ
- Формат: SSD M2

Программное обеспечение:

- Операционная система: Ubuntu 24.04 LTS
- IDE: Lunar Vim

Метод решения

Методика решения основана на использовании параллельных вычислений на GPU для классификации пикселей изображения.

На первом этапе загружаются входные данные, включая изображение в формате uchar4, представляющее пиксели в четырехканальном формате (RGB и дополнительный канал, используемый для хранения индекса класса). Также считываются координаты пикселей, относящихся к разным классам, чтобы вычислить средние значения цветов для каждого класса.

Для вычисления средних значений цветов используется метод суммирования значений каналов для каждого пикселя внутри класса с последующим делением на количество пикселей. Эти средние значения загружаются в константную память GPU (__constant__ float3 devAvg[32]), что позволяет быстро обращаться к ним во время вычислений.

На следующем этапе происходит копирование изображения в глобальную память GPU. Запускается CUDA-ядро, в котором каждому потоку назначается обработка отдельного пикселя изображения. Для каждого пикселя вычисляется евклидово расстояние до всех средних значений классов, и выбирается класс с минимальным расстоянием. Результат записывается в альфа-канал пикселя (data[i].w = j).

Производительность вычислений проверяется за счет варьирования конфигурации запуска ядра CUDA, где изменяется число блоков и потоков на блок. Измеряется время выполнения каждого варианта, после чего изображение с обновленной классификацией пикселей копируется обратно в оперативную память и записывается в выходной файл.

Описание программы

В программе реализованы две функции.

Первая функция, kernel, представляет собой CUDA-ядро, которое выполняется на графическом процессоре. Оно обрабатывает пиксели изображения, назначая каждому пикселю класс на основе ближайшего среднего цвета из предопределенного набора.

Каждый поток вычисляет индекс пикселя, который он должен обработать, используя posX = blockDim.x * blockIdx.x + threadIdx.x. Затем потоки двигаются с шагом shiftX = blockDim.x * gridDim.x, чтобы обработать все пиксели изображения. Для каждого пикселя вычисляется евклидово расстояние в цветовом пространстве между его цветом и средними значениями классов, хранящимися в константной памяти (devAvg). Вместо стандартного корня в расстоянии используется отрицательное квадратное отклонение, что позволяет избежать лишних вычислений. Если текущее расстояние больше найденного ранее (по сути, находим ближайший класс), обновляется максимальное значение и записывается соответствующий класс в альфа-канал (data[i].w).

```
__global__ void kernel(uchar4 *data, int width, int height, int nc)
{
   int posX = blockDim.x * blockIdx.x + threadIdx.x;
```

```
    int shiftX = blockDim.x * gridDim.x;

    for (int i = posX; i < width * height; i += shiftX)
    {
        float max = -FLT_MAX;

        for (int j = 0; j < nc; j++)
        {
            float cur = -(data[i].x - devAvg[j].x) * (data[i].x - devAvg[j].x) \
                        - (data[i].y - devAvg[j].y) * (data[i].y - devAvg[j].y) \
                        - (data[i].z - devAvg[j].z) * (data[i].z - devAvg[j].z);

            if (cur > max)
            {
                max = cur;
                data[i].w = j;
            }
        }
    }
}
```

Вторая функция, GetAvg, выполняется на центральном процессоре и вычисляет средний цвет каждого класса на основе предоставленных позиций пикселей. Сначала инициализируется массив средних значений (res), затем для каждого класса берутся пиксели, указанные в pos, их значения по каналам (R, G, B) суммируются, а затем усредняются, разделяя на количество пикселей. В результате получается вектор средних цветов, который затем используется в kernel для классификации пикселей.

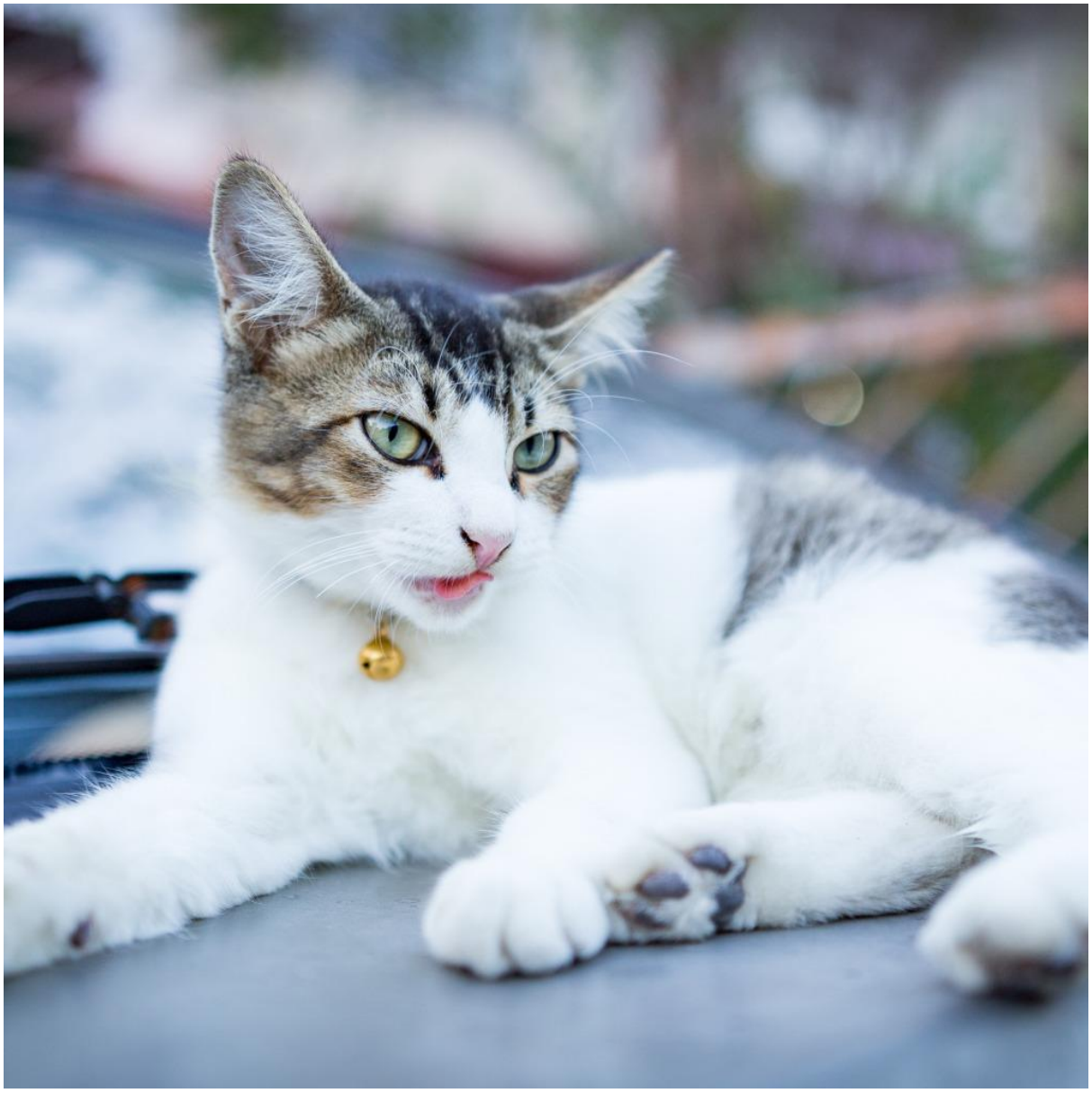
```
std::vector<float3> GetAvg(const std::vector<int>& np, std::vector<std::vector<std::pair<int, int>>>& pos, std::vector<uchar4>& data, int width)
{
    std::vector<float3> res(np.size());
    float3 acc;

    for (size_t i = 0; i < np.size(); i++)
    {
```

```
        acc = {0, 0, 0};

        for (size_t j = 0; j < np[i]; j++)
        {
            uchar4 cur = data[pos[i][j].first + pos[i][j].second * width];

            acc.x += cur.x;
            acc.y += cur.y;
            acc.z += cur.z;
        }

        acc.x /= np[i];
        acc.y /= np[i];
        acc.z /= np[i];

        res[i] = acc;
    }

    return res;
}
```

Результаты

В качестве входных данных были использованы два изображения: размерами 1024x1024 пикселя и 4096x4096 пикселей.

1. (1024x1024)



2. (4096x4096)

Замеры производительности программы:
Все измерения приведены в миллисекундах

Конфигурация ядра	Картинка №1 (1024x1024)	Картинка №2 (4096x4096)
<<<64, 64>>>	0.683284	6.48451
<<<128, 128>>>	0.070537	4.05735
<<<256, 256>>>	0.048413	3.53172
<<<512, 512>>>	0.043046	3.12962
<<<1024, 1024>>>	0.054255	3.19187

Выводы

Программа предназначена для классификации пикселей изображения на основе их цветовых характеристик с использованием параллельных вычислений на GPU. Алгоритм применим в задачах компьютерного зрения, таких как сегментация изображений, распознавание объектов и предобработка данных для машинного обучения.

Основной задачей алгоритма является определение класса каждого пикселя, исходя из минимального евклидова расстояния до заранее вычисленных средних значений классов. Это позволяет эффективно обрабатывать изображения с большой размерностью, используя ресурсы GPU.

Сложность программирования заключается в правильной организации вычислений на GPU, управлении памятью и выборе оптимальных параметров конфигурации сетки CUDA. Одной из проблем было определение подходящего размера блока и сетки потоков для максимальной производительности. Также важную роль сыграло использование константной памяти (__constant__), что позволило уменьшить задержки при доступе к данным.

Результаты тестирования показали, что время выполнения зависит от конфигурации сетки CUDA. С увеличением количества потоков на блок обработка ускоряется, но после определенного предела производительность перестает значительно расти из-за ограничений ресурсов GPU. Оптимальная конфигурация определяется экспериментально, исходя из размеров изображения и характеристик видеокарты.