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.Е. Аксенов*

Группа: *8О-408Б*

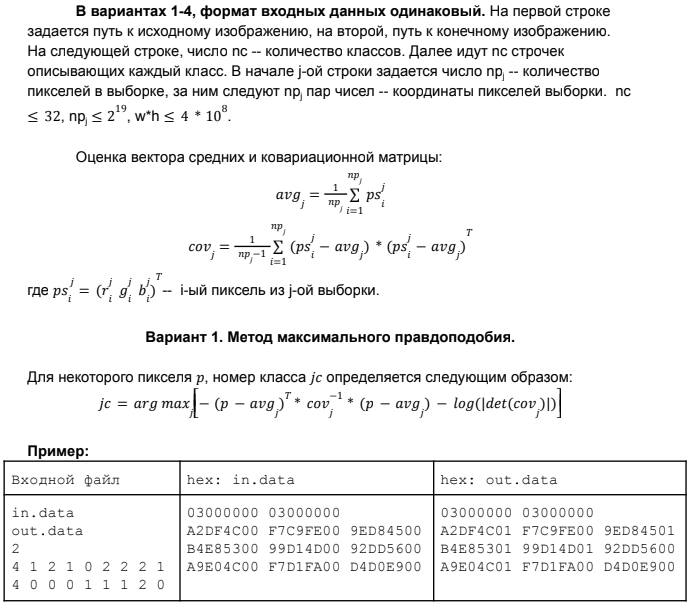
Преподаватели: К.Г. Крашенинников,

А.Ю. Морозов

Москва, 2023

**Условие**

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

Формат изображений соответствует формату описанному в лабораторной работе 2. Во всех вариантах, в результирующем изображении, на месте альфа-канала должен быть записан номер класса (кластера) к которому был отнесен соответствующий пиксель. Если пиксель можно отнести к нескольким классам, товыбирается класс с наименьшим номером.

**Программное и аппаратное обеспечение**

|  |  |
| --- | --- |
| Графический процессор | |
| GPU | NVIDIA RTX 2070 |
| Compute Capability | 7.5 |
| Объем видеопамяти | 8192 МБ |
| Тип видеопамяти | GDDR6 |
| Разделяемая память | 48 КБ |
| Константная память | 64 КБ |
| Число регистров в блоке | 65636 |
| Максимальное число блоков | 16 |
| Максимальное число нитей | 1024 |
| Количество мультипроцессоров | 34 |
| Центральный процессор | |
| СPU | Intel Core i5 9600K |
| Число физических ядер | 6 |
| Максимальное число потоков | 6 |
| Базовая частота | 3.6 GHz |
| Частота Turbo Boost | 4.5 GHz |
| Объем кэша L3 | 9 МБ |
| Оперативная память | |
| Тип оперативной памяти: | DDR4 |
| Частота памяти | 2666MHz |
| Объём | 32 ГБ |
| Постоянная память | |
| Тип постоянной памяти | SSD |
| Объём постоянной памяти | 1024ГБ |
| Программное обеспечение | |
| Операционная система | Microsoft Windows 11 |
| Версия операционной системы | 2004 |
| Среда разработки | Microsoft Visual Studio 2022 |
| Версия среды разработки | 16.7.3 |
| NVIDIA CUDA Toolkit | 12.2 |

**Метод решения**

1. Считываем параметры входных данных, т.е. имена входных и выходных файлов, количество классов.
2. Для каждого класса считываем точки, которые ему принадлежат, в то же время производим расчёт центра масс для данного класса.
3. После того, как выполнен расчёт центров масс и все точки классов находятся в оперативной памяти, производим расчёт матрицы ковариации для каждого класса.
4. На данном этапе все требуемые константы подсчитаны на центральном процессоре, массив точек больше не требуется и его можно освободить.
5. Для подсчитанных констант в глобальной области программы определена константная память фиксированного размера. Требуется выполнить копирование всех вычисленных констант в данную область памяти при помощи функции cudaMemcpyToSymbol. Использование константной памяти для выполнения такого типа вычислений позволит заметно увеличить скорость доступа за счёт аппаратного кэширования и рассылки фрагментов константных данных.
6. Выполняется копирование входного изображения в память устройства.
7. Выполняется вызов ядра с оптимальными параметрами сетки и блоков. Каждый поток вызывает функцию классификации и определяет наиболее подходящий класс для обрабатываемого пикселя. Для классификации используется функция подсчёта расстояния Махаланобиса, помеченная классификатором \_\_device\_\_.
8. Выполняется копирование изображения обратно на хост.
9. Освобождается память на устройстве.
10. Выводим результаты в файл в соответствии с требованиями.
11. Освобождаем память на хосте.
12. Завершаем работу программы.

**Описание программы**

Программа реализована в виде одного файла с исходным кодом. В ней имеется реализация одного единственного ядра, которая сопровождается квалификатором \_\_global\_\_. Для того, чтобы избежать дублирования фрагментов кода, алгоритм классификации вынесен в отдельную функцию, помеченную квалификатором \_\_device\_\_. Просчёт требуемых констант осуществляется на центральном процессоре, время на данные операции не учитывается. Вычисления матрицы обратной ковариационной выполняется в соответствии с алгоритмом обращения с использованием союзной матрицы. Для данного алгоритма требуется вычисление детерминанта, с этой целью реализован алгоритм вычисления, использующий правило треугольников. Константная память определена в глобальной области и помечена квалификатором \_\_constant\_\_. Изображение представляет из себя массив элементов типа uchar4, в котором можно хранить 1 байтовые компоненты rgb вместе с альфа каналом. Для подсчёта времени работы устройства используются события вместе с синхронизацией по завершению выполнения.

\_\_global\_\_ void kern(uchar4\* const image\_bytes,

    const int32\_t pixel\_count, const int32\_t nc) {

    const int32\_t block\_index = gridDim.x \* blockIdx.y + blockIdx.x;        // линейный индекс блока относительно сетки

    const int32\_t thread\_index = blockDim.x \* threadIdx.y + threadIdx.x;    // линейный индекс потока относительно блока

    const int32\_t block\_size = blockDim.x \* blockDim.y;                     // размер блока

    const int32\_t grid\_size = gridDim.x \* gridDim.y \* block\_size;           // размер сетки в потоках

    for(int32\_t k = block\_index \* block\_size + thread\_index;                // считаем номер класса для пикселей

                            k < pixel\_count; k += grid\_size)  {

        double max\_val = eval(image\_bytes[k], dev\_cvals[0]);

        unsigned char& max\_idx = image\_bytes[k].w; max\_idx = 0;

        for(int32\_t j = 1; j < nc; ++j){

            double val = eval(image\_bytes[k], dev\_cvals[j]);

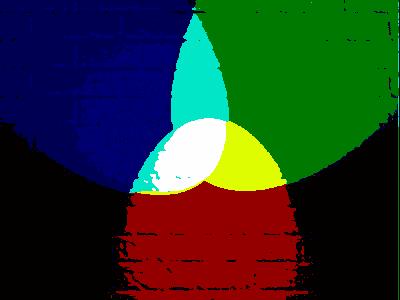
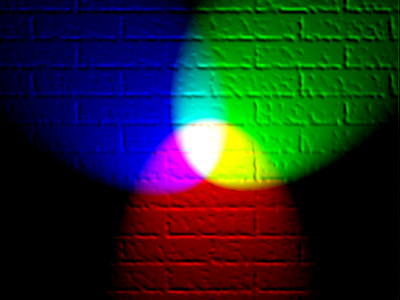
            if (val > max\_val) {max\_idx = j; max\_val = val;}

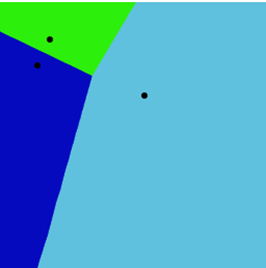
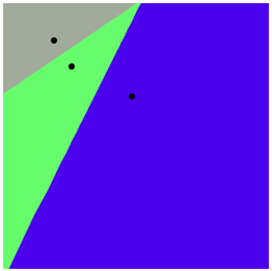
        }

    }

}

**Результаты**





|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размеры изображения | Количество кластеров | Ядро | Время GPU мс | Время CPU  мс |
| 50 000 000 | 32 | (32 32)(32 32) | 437.87 | 34079 |
| 100 000 000 | 32 | (32 32) (32 32) | 877.054 | 68517 |
| 150 000 000 | 32 | (32 32) (32 32) | 1324.72 | 101050 |
| 200 000 000 | 32 | (32 32) (32 32) | 1792.6 | 137107 |
| 250 000 000 | 32 | (32 32) (32 32) | 2205.61 | 170940 |
| 300 000 000 | 32 | (32 32) (32 32) | 2693.58 | 203292 |
| 350 000 000 | 32 | (32 32) (32 32) | 3137.89 | 237984 |
| 400 000 000 | 32 | (32 32) (32 32) | 3585.66 | 281617 |

|  |  |  |  |
| --- | --- | --- | --- |
| Размеры изображения | Количество кластеров | Ядро  (сетка) (блок) | Время GPU  мс |
| 200 000 000 | 32 | (32 1)(32 32) | 1849.44 |
| 200 000 000 | 32 | (32 8)(32 32) | 1845.93 |
| 200 000 000 | 32 | (32 16)(32 32) | 1846.21 |
| 200 000 000 | 32 | (32 32)(32 32) | 1793.38 |
| 200 000 000 | 32 | (32 64)(32 32) | 1755.29 |
| 200 000 000 | 32 | (32 32)(32 1) | 1830.03 |
| 200 000 000 | 32 | (32 32)(32 8) | 1787.21 |
| 200 000 000 | 32 | (32 32)(32 16) | 1787.9 |

**Выводы**

Расстояние максимального правдоподобия широко используется в кластерном анализе и алгоритмах классификации. При реализации алгоритма особых проблем не возникло. Константная память в виду структуры решаемой задачи позволила оптимизировать обращение к глобальной памяти путём кэширования и рассылки фрагментов данных. Это стало возможным благодаря тому, что все нити в пределах одного блока обращаются непосредственно к одной и той же области памяти. Стоит отметить, что при неправильной организации доступа к такой памяти будут наблюдаться продолжительные задержки, к тому же стоит учитывать, что объём такой памяти ограничен моделью устройства. В идеале стоит адаптировать алгоритм для того, чтобы он смог работать как на старых устройствах, так и на новых.