МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии 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. О. Тояков

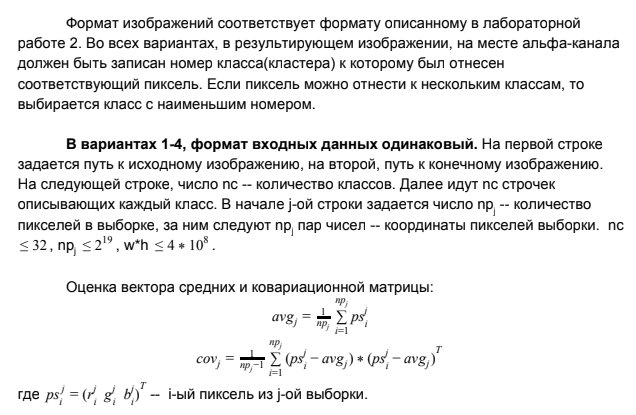
Группа: М8О-407Б-18

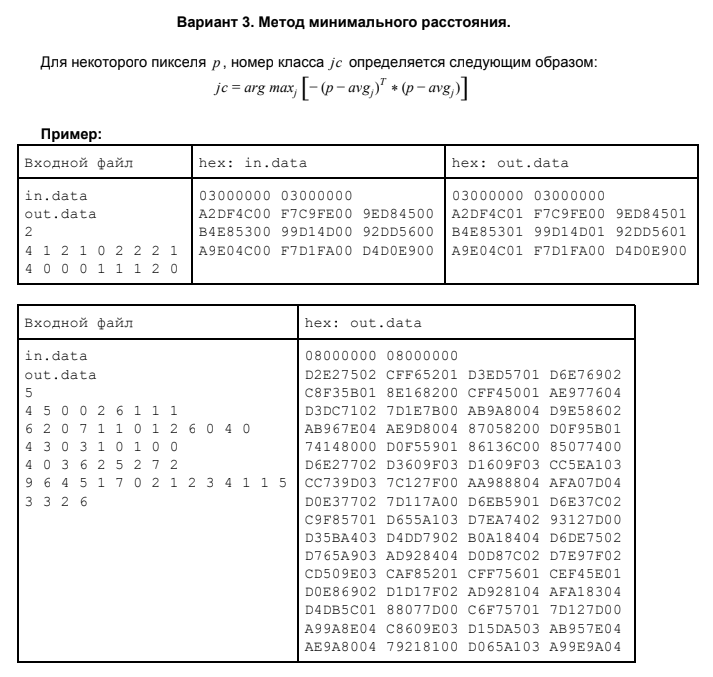
Преподаватели: К. Г. Крашенинников,

А. Ю. Морозов

# Условие

**Цель работы:** научиться использовать GPU для классификации и кластеризации изображений. Использование константной памяти.





# Программное и аппаратное обеспечение

Device: GeForce MX250

Размер глобальной памяти: 3150381056

Размер константной памяти : 65536

Размер разделяемой памяти: 49152

Регистров на блок: 32768

Максимум потоков на блок: 1024

Количество мультипроцессоров : 3

OS: Linux Ubuntu 18.04

Редактор: VSCode

Компилятор: nvcc версии 11.4 (g++ версии 7.5.0)

# Метод решения

Для кластеризации изображений методом минимального расстояния необходимо сначала посчитать avg по формуле, приведённой выше, занести значения в константную память для работы на GPU. Затем на девайсе произвести вычисление подходящего номера класса и записать его в альфа-канал каждого пикселя.

# Описание программы

Макрос **CSC**отвечает за отслеживание ошибок в функциях cuda, поэтому все cuda-вызовы оборачиваются в него и при cudaError\_t != cudaSuccess выводится сообщение об ошибке.

\_\_global\_\_ void kernel() – функция на GPU, в которой происходит определение класса пикселя.

\_\_device\_\_ double get\_min\_dist() – функция на GPU, которая для каждого класса просчитывает минимальное расстояние по формуле.

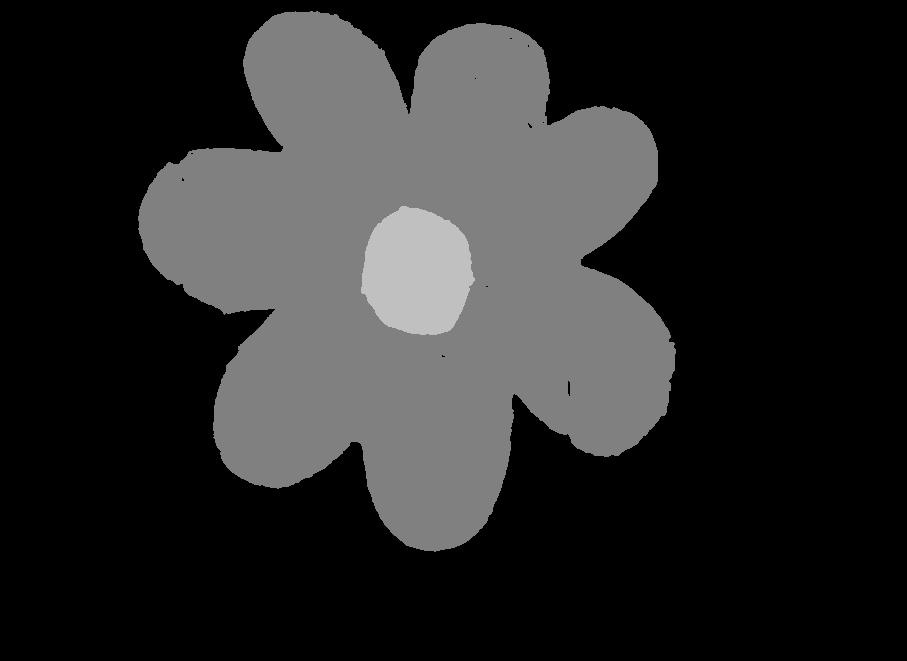
int main() ­– отвечает за ввод, расчёт и перенос данных avg в константный массив, передачу данных в kernel и вывод.

struct point – структура данных, хранящая пару интовых значений – координаты пикселя в выборке.

\_\_constant\_\_ double dev\_avg[32][3] – константный массив для работы с матожиданием на GPU, где 32 – максимальное количество классов, 3 – цветовые компоненты RGB.

# Пример работы программы

Исходное изображение: 

После классификации на три класса: 

# тесты производительности

Количество кластеров и количество пикселей в выборке во всех тестах было небольшим и хорошо подобранным под изображение.

Работа на GPU:

|  |  |
| --- | --- |
| Тест: | Результат: |
| 232 \* 218 | kernel = «<1, 32»>, time = 1.554400 kernel = «<1, 64»>, time = 0.773952 kernel = «<1, 128»>, time = 0.480256 kernel = «<1, 256»>, time = 0.385024 kernel = «<1, 512»>, time = 0.369664 kernel = «<1, 1024»>, time = 0.368640 kernel = «<2, 32»>, time = 0.776192 kernel = «<2, 64»>, time = 0.475136 kernel = «<2, 128»>, time = 0.242688 kernel = «<2, 256»>, time = 0.193536 kernel = «<2, 512»>, time = 0.188416 kernel = «<2, 1024»>, time = 0.191488 kernel = «<4, 32»>, time = 0.482304 kernel = «<4, 64»>, time = 0.243712 kernel = «<4, 128»>, time = 0.194560 kernel = «<4, 256»>, time = 0.187392 kernel = «<4, 512»>, time = 0.195584 kernel = «<4, 1024»>, time = 0.190464 kernel = «<8, 32»>, time = 0.248832 kernel = «<8, 64»>, time = 0.160768 kernel = «<8, 128»>, time = 0.144384 kernel = «<8, 256»>, time = 0.144384 kernel = «<8, 512»>, time = 0.151552 kernel = «<8, 1024»>, time = 0.184320 kernel = «<16, 32»>, time = 0.198656 kernel = «<16, 64»>, time = 0.144384 kernel = «<16, 128»>, time = 0.145408 kernel = «<16, 256»>, time = 0.146432 kernel = «<16, 512»>, time = 0.143360 kernel = «<16, 1024»>, time = 0.141312 kernel = «<32, 32»>, time = 0.143360 kernel = «<32, 64»>, time = 0.142336 kernel = «<32, 128»>, time = 0.131072 kernel = «<32, 256»>, time = 0.131072 kernel = «<32, 512»>, time = 0.130048 kernel = «<32, 1024»>, time = 0.130048 kernel = «<64, 32»>, time = 0.141312 kernel = «<64, 64»>, time = 0.135168 kernel = «<64, 128»>, time = 0.136192 kernel = «<64, 256»>, time = 0.137216 kernel = «<64, 512»>, time = 0.131072 kernel = «<64, 1024»>, time = 0.133120 kernel = «<128, 32»>, time = 0.131072 kernel = «<128, 64»>, time = 0.133120 kernel = «<128, 128»>, time = 0.125952 kernel = «<128, 256»>, time = 0.126976 kernel = «<128, 512»>, time = 0.125952 kernel = «<128, 1024»>, time = 0.134144 kernel = «<256, 32»>, time = 0.132096 kernel = «<256, 64»>, time = 0.130048 kernel = «<256, 128»>, time = 0.130048 kernel = «<256, 256»>, time = 0.135168 kernel = «<256, 512»>, time = 0.132096 kernel = «<256, 1024»>, time = 0.135136 kernel = «<512, 32»>, time = 0.128000 kernel = «<512, 64»>, time = 0.128000 kernel = «<512, 128»>, time = 0.128000 kernel = «<512, 256»>, time = 0.132096 kernel = «<512, 512»>, time = 0.137216 kernel = «<512, 1024»>, time = 0.145408 kernel = «<1024, 32»>, time = 0.129024 kernel = «<1024, 64»>, time = 0.126976 kernel = «<1024, 128»>, time = 0.129024 kernel = «<1024, 256»>, time = 0.132096 kernel = «<1024, 512»>, time = 0.143360 kernel = «<1024, 1024»>, time = 0.163840 |
| 907 \* 661 | kernel = «<1, 32»>, time = 17.700865 kernel = «<1, 64»>, time = 8.872960 kernel = «<1, 128»>, time = 4.550560 kernel = «<1, 256»>, time = 3.043328 kernel = «<1, 512»>, time = 3.105792 kernel = «<1, 1024»>, time = 2.629632 kernel = «<2, 32»>, time = 11.622400 kernel = «<2, 64»>, time = 4.760576 kernel = «<2, 128»>, time = 3.495872 kernel = «<2, 256»>, time = 1.458176 kernel = «<2, 512»>, time = 1.312768 kernel = «<2, 1024»>, time = 1.313792 kernel = «<4, 32»>, time = 6.913024 kernel = «<4, 64»>, time = 2.550784 kernel = «<4, 128»>, time = 1.474560 kernel = «<4, 256»>, time = 1.310720 kernel = «<4, 512»>, time = 1.316864 kernel = «<4, 1024»>, time = 1.313792 kernel = «<8, 32»>, time = 4.243456 kernel = «<8, 64»>, time = 1.305600 kernel = «<8, 128»>, time = 1.067008 kernel = «<8, 256»>, time = 0.898048 kernel = «<8, 512»>, time = 0.897024 kernel = «<8, 1024»>, time = 1.190912 kernel = «<16, 32»>, time = 1.359872 kernel = «<16, 64»>, time = 0.911360 kernel = «<16, 128»>, time = 0.905216 kernel = «<16, 256»>, time = 0.899072 kernel = «<16, 512»>, time = 0.898048 kernel = «<16, 1024»>, time = 0.896000 kernel = «<32, 32»>, time = 0.910336 kernel = «<32, 64»>, time = 0.898048 kernel = «<32, 128»>, time = 0.826368 kernel = «<32, 256»>, time = 0.898048 kernel = «<32, 512»>, time = 0.821248 kernel = «<32, 1024»>, time = 0.894976 kernel = «<64, 32»>, time = 0.823296 kernel = «<64, 64»>, time = 0.822272 kernel = «<64, 128»>, time = 0.823296 kernel = «<64, 256»>, time = 0.823296 kernel = «<64, 512»>, time = 0.833536 kernel = «<64, 1024»>, time = 0.825344 kernel = «<128, 32»>, time = 0.826368 kernel = «<128, 64»>, time = 0.825344 kernel = «<128, 128»>, time = 0.805888 kernel = «<128, 256»>, time = 0.823296 kernel = «<128, 512»>, time = 0.824320 kernel = «<128, 1024»>, time = 0.807936 kernel = «<256, 32»>, time = 0.823296 kernel = «<256, 64»>, time = 0.849920 kernel = «<256, 128»>, time = 0.828416 kernel = «<256, 256»>, time = 0.826368 kernel = «<256, 512»>, time = 0.825344 kernel = «<256, 1024»>, time = 0.887808 kernel = «<512, 32»>, time = 0.836608 kernel = «<512, 64»>, time = 0.838656 kernel = «<512, 128»>, time = 0.822240 kernel = «<512, 256»>, time = 0.803840 kernel = «<512, 512»>, time = 0.817152 kernel = «<512, 1024»>, time = 0.815104 kernel = «<1024, 32»>, time = 0.811008 kernel = «<1024, 64»>, time = 0.808960 kernel = «<1024, 128»>, time = 0.808960 kernel = «<1024, 256»>, time = 0.806912 kernel = «<1024, 512»>, time = 0.810976 kernel = «<1024, 1024»>, time = 0.823296 |
| 1280 \* 1024 | kernel = «<1, 32»>, time = 44.448769 kernel = «<1, 64»>, time = 25.694208 kernel = «<1, 128»>, time = 15.100928 kernel = «<1, 256»>, time = 12.765184 kernel = «<1, 512»>, time = 12.653536 kernel = «<1, 1024»>, time = 13.502464 kernel = «<2, 32»>, time = 29.296640 kernel = «<2, 64»>, time = 16.448511 kernel = «<2, 128»>, time = 7.503872 kernel = «<2, 256»>, time = 7.284736 kernel = «<2, 512»>, time = 4.300800 kernel = «<2, 1024»>, time = 4.456448 kernel = «<4, 32»>, time = 16.421888 kernel = «<4, 64»>, time = 7.274400 kernel = «<4, 128»>, time = 5.628928 kernel = «<4, 256»>, time = 4.334592 kernel = «<4, 512»>, time = 5.594112 kernel = «<4, 1024»>, time = 4.888576 kernel = «<8, 32»>, time = 6.287360 kernel = «<8, 64»>, time = 5.612544 kernel = «<8, 128»>, time = 3.344384 kernel = «<8, 256»>, time = 3.628032 kernel = «<8, 512»>, time = 4.524032 kernel = «<8, 1024»>, time = 3.380224 kernel = «<16, 32»>, time = 4.037632 kernel = «<16, 64»>, time = 4.462592 kernel = «<16, 128»>, time = 3.798016 kernel = «<16, 256»>, time = 3.236864 kernel = «<16, 512»>, time = 3.230720 kernel = «<16, 1024»>, time = 3.231744 kernel = «<32, 32»>, time = 3.234816 kernel = «<32, 64»>, time = 4.306944 kernel = «<32, 128»>, time = 2.966528 kernel = «<32, 256»>, time = 3.138560 kernel = «<32, 512»>, time = 2.981888 kernel = «<32, 1024»>, time = 3.347456 kernel = «<64, 32»>, time = 4.084736 kernel = «<64, 64»>, time = 3.259392 kernel = «<64, 128»>, time = 2.977792 kernel = «<64, 256»>, time = 4.034560 kernel = «<64, 512»>, time = 3.101696 kernel = «<64, 1024»>, time = 4.370432 kernel = «<128, 32»>, time = 2.974720 kernel = «<128, 64»>, time = 4.236288 kernel = «<128, 128»>, time = 3.265536 kernel = «<128, 256»>, time = 2.985984 kernel = «<128, 512»>, time = 3.079168 kernel = «<128, 1024»>, time = 4.429824 kernel = «<256, 32»>, time = 3.061760 kernel = «<256, 64»>, time = 2.909184 kernel = «<256, 128»>, time = 4.122624 kernel = «<256, 256»>, time = 3.870720 kernel = «<256, 512»>, time = 2.794496 kernel = «<256, 1024»>, time = 3.994624 kernel = «<512, 32»>, time = 3.228672 kernel = «<512, 64»>, time = 3.354624 kernel = «<512, 128»>, time = 2.749440 kernel = «<512, 256»>, time = 4.441088 kernel = «<512, 512»>, time = 2.918400 kernel = «<512, 1024»>, time = 3.402752 kernel = «<1024, 32»>, time = 2.766848 kernel = «<1024, 64»>, time = 2.744320 kernel = «<1024, 128»>, time = 4.002688 kernel = «<1024, 256»>, time = 3.672064 kernel = «<1024, 512»>, time = 3.733504 kernel = «<1024, 1024»>, time = 2.751488 |

Работа на CPU:

|  |  |
| --- | --- |
| Тест: | Результат: |
| 232 \* 218 | 6.9 |
| 907 \* 661 | 55.774 |
| 1280 \* 1024 | 91.158 |

# Выводы

Сделав лабораторную работу № 3 я познакомился с константной памятью в CUDA и снова приблизился к методам классификации. Для начала хочу сказать, что этот вид памяти довольно быстрый, но при этом на GPU доступно только считывание из этой памяти. Также чтобы перенести данные с хоста на девайс нужно использовать функцию cudaMemcpyToSymbol.

Я реализовал метод минимального расстояния, который на мой взгляд работает гораздо хуже, чем метод Махаланобиса и тем более метод максимального правдоподобия за счёт того, что в них используется матрица ковариаций. В целом, задачи классификации удобно распараллеливать, поэтому не удивительно, что результаты на GPU гораздо лучше, чем на CPU. Также результат сильно зависит от количества кластеров и пикселей выборки и цветовой палитры изображения.