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

Москва, 2021

УСЛОВИЕ

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

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, то выбирается класс с наименьшим номером.

В вариантах 1-4, формат входных данных одинаковый. На первой строке задается путь к исходному изображению, на второй, путь к конечному изображению. На следующей строке, число nc -- количество классов. Далее идут nc строчек описывающих каждый класс. В начале j-ой строки задается число np_j -- количество пикселей в выборке, за ним следуют np_j пар чисел -- координаты пикселей выборки. $nc \leq 32$, $np_j \leq 2^{19}$, $w*h \leq 4 * 10^8$.

Оценка вектора средних и ковариационной матрицы:

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

$$cov_j = \frac{1}{np_j - 1} \sum_{i=1}^{np_j} (ps_i^j - avg_j) * (ps_i^j - avg_j)^T$$

где $ps_i^j = (r_i^j \; g_i^j \; b_i^j)^T$ -- i-ый пиксель из j-ой выборки.

Вариант 3. Метод минимального расстояния.

Для некоторого пикселя p, номер класса jc определяется следующим образом:

$$jc = arg\ max_j \left[-(p - avg_j)^T * (p - avg_j) \right]$$

Пример:

Входной файл	hex: in.data	hex: out.data
in.data out.data 2 4 1 2 1 0 2 2 2 1 4 0 0 0 1 1 1 2 0	03000000 03000000 A2DF4C00 F7C9FE00 9ED84500 B4E85300 99D14D00 92DD5600 A9E04C00 F7D1FA00 D4D0E900	03000000 03000000 A2DF4C01 F7C9FE00 9ED84501 B4E85301 99D14D01 92DD5601 A9E04C01 F7D1FA00 D4D0E900

Входной файл	hex: out.data
in.data out.data 5 4 5 0 0 2 6 1 1 1 6 2 0 7 1 1 0 1 2 6 0 4 0 4 3 0 3 1 0 1 0 0 4 0 3 6 2 5 2 7 2 9 6 4 5 1 7 0 2 1 2 3 4 1 1 5 3 3 2 6	08000000 08000000 D2E27502 CFF65201 D3ED5701 D6E76902 C8F35B01 8E168200 CFF45001 AE977604 D3DC7102 7D1E7B00 AB9A8004 D9E58602 AB967E04 AE9D8004 87058200 D0F95B01 74148000 D0F55901 86136C00 85077400 D6E27702 D3609F03 D1609F03 CC5EA103 CC739D03 7C127F00 AA988804 AFA07D04 D0E37702 7D117A00 D6EB5901 D6E37C02 C9F85701 D655A103 D7EA7402 93127D00 D35BA403 D4DD7902 B0A18404 D6DE7502 D765A903 AD928404 D0D87C02 D7E97F02 CD509E03 CAF85201 CFF75601 CEF45E01 D0E86902 D1D17F02 AD928104 AFA18304 D4DB5C01 88077D00 C6F75701 7D127D00 A99A8E04 C8609E03 D15DA503 AB957E04 AE9A8004 79218100 D065A103 A99E9A04

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.

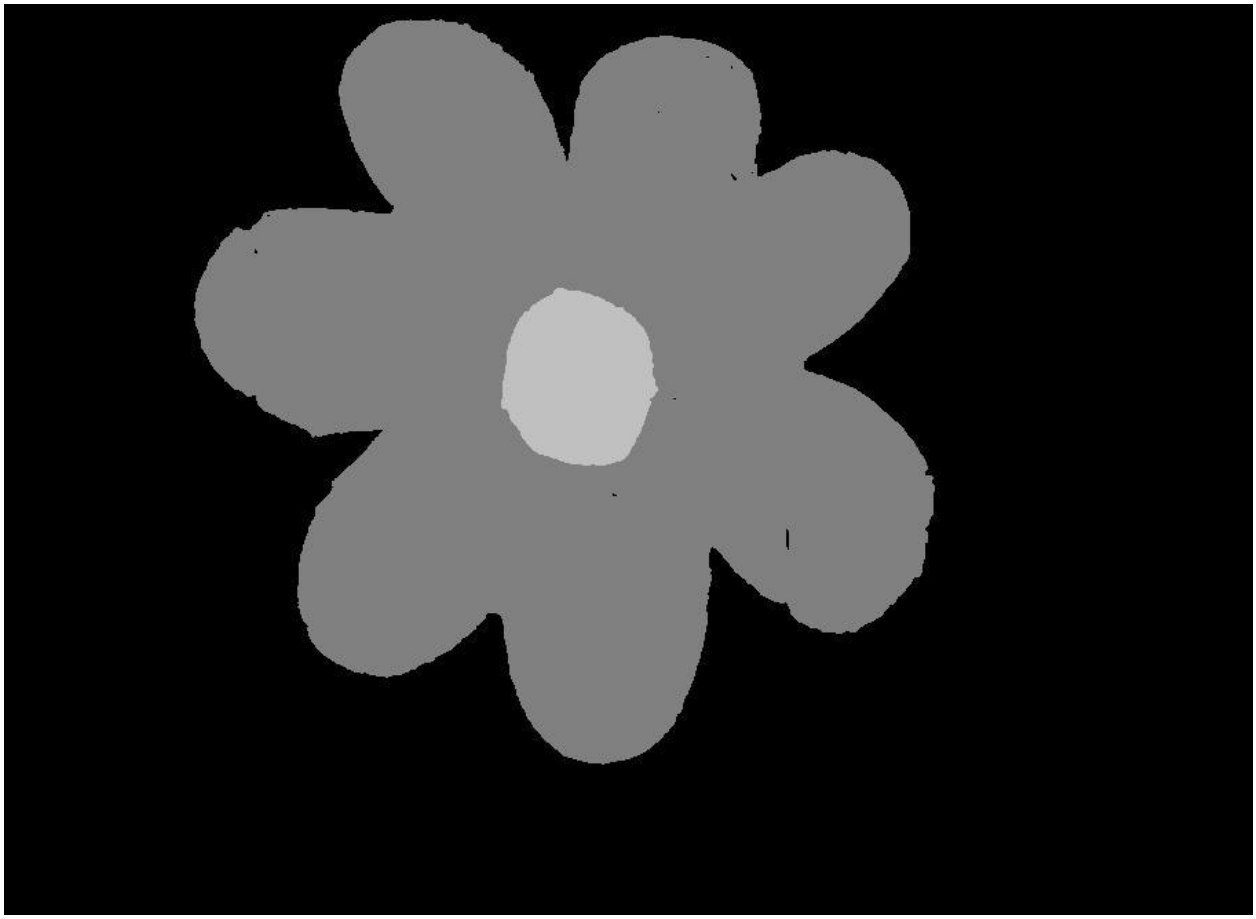
struct point – структура данных, хранящая пару интовых значений – координаты пикселя в выборке.

__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.