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»

Изучение технологии CUDA

Выполнил: А. О. Дубинин
Группа: 8О-407Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2020

Условие

Цель работы:

Научиться использовать GPU для классификации и кластеризации изображений.
Использование константной памяти.

Вариант 1. Метод максимального правдоподобия.

Для некоторого пикселя p , номер класса j_c с определяется следующим образом:

$$j_c = argmax_j[-(p-avg_j)^T * cov_j^{-1} * (p-avg_j) - \log(||cov_j||)]$$

Программное и аппаратное обеспечение

GeForce 940MX

Compute capability:	5.0
Dedicated video memory:	4096 MB
shared memory per block:	49152 bytes
constant memory:	65536 bytes
Total number of registers available per block:	65536
Maximum number of threads per multiprocessor:	2048
Maximum number of threads per block:	1024
(3) Multiprocessors, (128) CUDA Cores/MP:	384 CUDA Cores

Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz

Architecture:	x86_64
Byte Order:	Little Endian
CPU(s):	4
Thread(s) per core:	2
Core(s) per socket:	2
CPU MHz: CPU max MHz: CPU min MHz:	713.848 3100,0000 400,0000
L1d cache: L1i cache: L2 cache: L3 cache:	64 KiB 64 KiB 512 KiB 3 MiB

RAM	8GiB SODIMM DDR4 Synchronous Unbuffered (Unregistered) 2400 MHz (0,4 ns)

SSD(SPCC_M.2_SSD)	223,6G
HDD(ST1000LM035-1RK172)	931,5G

OS: Ubuntu 20.04 focal
IDE: jetbrains clion
compiler: nvcc

Метод решения

Посчитал все нужные значения для формулы для всех классов на хосте. Эти значения константны, но используются для классификации пикселя. Записал значения в константную память. На device'e выполнил пробег по всем пикселям и классифицировал все пиксели на классы, значения класса записал в альфа канал.

Описание программы

Подсчет значений у меня вышел тривиальным, так как у нас размеры матриц всегда одинаковые, то я воспользовался методом хардкода, подсчитав все значения напрямую. В kernel я не ушел далеко от этой концепции и посчитал перемножение матриц зная заранее размер матриц, брав заранее посчитанные значения оценок вектора средних и обратной ковариационной матрицы и детерминанта этой матрицы из константной памяти.

```
const int MAXX = 1e8;
__global__ void kernel(uchar4 *data, int w, int h, int nc) {
    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    int idy = blockDim.y * blockIdx.y + threadIdx.y;
    int offsetx = blockDim.x * gridDim.x;
    int offsety = blockDim.y * gridDim.y;
    int x, y, i, j, k;
    uchar4 ps;

    for (y = idy; y < h; y += offsety) {
        for (x = idx; x < w; x += offsetx) {
            ps = data[y * w + x];

            double mx = -MAXX;
            int idx = -1;
            for (i = 0; i < nc; ++i) {

                int diff[3];
                diff[0] = ps.x - avg_dev[i].x;
                diff[1] = ps.y - avg_dev[i].y;
                diff[2] = ps.z - avg_dev[i].z;

                double tmp[3];
                for (j = 0; j < 3; ++j) {
                    tmp[j] = 0;
                    for (k = 0; k < 3; ++k) {
                        tmp[j] += (diff[k] * cov_inv_dev[i][k][j]);
                    }
                }
                double ans = 0;
                for (j = 0; j < 3; ++j) {
                    ans += (tmp[j] * diff[j]);
                }
                ans = -ans - log(abs(dets_dev[i]));

                if (ans > mx) {
                    mx = ans;
                    idx = i;
                }
            }
            data[y * w + x].w = idx;
        }
    }
}
```

Результаты

Замеры времени производятся с помощью cudaEventElapsedTime, как было показано на лекции**.**

1.

	Небольшой тест	Средний тест	Большой тест
<<<1, 16>>>, <<<1, 16>>>	8.72	138.83	2998.12
<<<16, 16>>>, <<<16, 16>>>	2.45	36.8	932.78
<<<1, 32>>>, <<<1, 32>>>	8.59	52.16	1110.91
<<<32, 32>>>, <<<32, 32>>>	2.57	36.68	935.69
<<<1, 64>>>, <<<1, 64>>>	8.28	48.71	1029.78
<<<64, 64>>>, <<<64, 64>>>	8.01	48.64	1064.8
<<<1, 128>>>, <<<1, 128>>>	8.01	49.22	1031.7
<<<1, 128>>>, <<<32, 32>>>	3.1	39.5	928.05
<<<128, 128>>>, <<<32, 32>>>	3.13	37.23	926.72
<<<256, 256>>>, <<<32,32>>>	6.29	47.29	975.4
<<<1, 512>>>, <<<1, 512>>>	9.5	92.34	1179.44
<<<512, 512>>>, <<<32, 32>>>	13.6	48.03	938.35

2.

	Небольшой тест	Средний тест	Большой тест
GPU<<<16, 16>>>, <<<16, 16>>>	2.97	45.27	987.12
CPU	38.42	592.67	15926.05

Выводы

Данная лабораторная работа показывает ещё одно полезное применение cuda. Классификация как мы видим по показателем выполняется гораздо быстрее на gpu, нежеле на cpu. Данный алгоритм имеет широкое применение в машинном обучении. У меня возникла сложность с тем, что в формуле очень много подсчетов и шанс сделать ошибку был велик, поэтому сначала я написал все на cpu и отлаживал каждый подсчет в формуле.