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: К.О. Вахрамян
Группа: 8О-406Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2021

Условие

Кратко описывается задача:

1. Цель работы.

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

2. Вариант задания.

Вариант 1. Метод максимального правдоподобия.

Программное и аппаратное обеспечение

GPU:

--- General Information for device ---
Name: NVIDIA GeForce GTX 1650
Compute capability: 7.5
Clock rate: 1560000
Device copy overlap: Enabled
Kernel execution timeout : Enabled
--- Memory Information for device ---
Total global mem: 4100521984
Total constant Mem: 65536
Max mem pitch: 2147483647
Texture Alignment: 512
--- MP Information for device ---
Multiprocessor count: 16
Shared mem per mp: 49152
Registers per mp: 65536
Threads in warp: 32
Max threads per block: 1024
Max thread dimensions: (1024, 1024, 64)
Max grid dimensions: (2147483647, 65535, 65535)

CPU:

Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
Address sizes: 39 bits physical, 48 bits virtual
CPU(s): 8
On-line CPU(s) list: 0-7
Thread(s) per core: 2
Core(s) per socket: 4
Socket(s): 1
NUMA node(s): 1
Vendor ID: GenuineIntel
CPU family: 6
Model: 158
Model name: Intel(R) Core(TM) i5-9300HF CPU @ 2.40GHz

Stepping: 13
CPU MHz: 1274.759
CPU max MHz: 2400.0000
CPU min MHz: 800.0000
BogoMIPS: 4800.00
Virtualization: VT-x
L1d cache: 128 KiB
L1i cache: 128 KiB
L2 cache: 1 MiB
L3 cache: 8 MiB

OS:
Linux Mint 20

Compiler:
nvcc

Code Editor:
VS Code

Метод решения

Для каждого класса вычислить вектор средних:

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

ковариационную матрицу:

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

её определитель и обратную матрицу.
Далее на устройстве для каждого пикселя определить класс по формуле:

$$arg\,max_j [-(p - avg_j)^T * cov_j^{-1} * (p - avg_j) - \log(|det(cov_j)|)]$$

Соответствующий индекс занести в альфа канал.

Описание программы

Вычисление вектора средних:

```
ans->avg = make_float4(0,0,0,0);
for (int i = 0; i < np; i++) {
uchar4 p = data[w * v[i].y + v[i].x];
ans->avg.x += p.x;
ans->avg.y += p.y;
ans->avg.z += p.z;
}
ans->avg.x /= np;
ans->avg.y /= np;
ans->avg.z /= np;
```

Ковариационной матрицы:

```
double cov[3][3] = {0.};
for (int i = 0; i < np; i++) {
uchar4 p = data[w * v[i].y + v[i].x];
double c[3] = {0.};
c[0] = p.x - ans->avg.x;
c[1] = p.y - ans->avg.y;
c[2] = p.z - ans->avg.z;

for (int k = 0; k < 3; k++)
for (int j = 0; j < 3; j++)
cov[k][j] += c[k] * c[j];
}
for (int k = 0; k < 3; k++)
for (int j = 0; j < 3; j++)
cov[k][j] /= np - 1;
```

Её определителя:

```
ans->det_cov = 0;
for (int i = 0; i < 3; i++) {
ans->det_cov += (cov[0][i] * cov[1][mod(i + 1)] * cov[2][mod(i + 2)] -
cov[0][mod(i + 2)] * cov[1][mod(i + 1)] * cov[2][i]);
}
```

И обратной к ковариационной матрице:

```
for (int i = 0; i < 3; i++)
for (int j = 0; j < 3; j++)
res[i][j] = ( cov[mod(j + 1)][mod(i + 1)] * cov[mod(j + 2)][mod(i + 2)] -
cov[mod(j + 1)][mod(i + 2)] * cov[mod(j + 2)][mod(i + 1)] ) /det;
```

Для каждого класса происходит на CPU.

Затем эти данные копируются в константную память:

```
CSC(cudaMemcpyToSymbol(features, host_features, sizeof(AvgCov) * nc));
```

После происходит вызов ядра:

```
kernel<<<blocks,threads>>>(dev_data, w, h, nc);
```

В ядре для каждого пикселя ищется максимум функции D(f):

```
for (int y = idy; y < h; y += offsety) {
for (int x = idx; x < w; x += offsetx) {
uchar4 p = data[y * w + x];
double max = D(p, &features[0]);
int arg = 0;
for (int class_n = 1; class_n < nc; class_n++) {
```

```
double d = D(p, &features[class_n]);
if (d > max) {
max = d;
arg = class_n;
}
}
data[y * w + x].w = arg;
}
```

Где D(f) определяется как:

$$D(f) = -(p - avg_j)^T * cov_j^{-1} * (p - avg_j) - \log(|det(cov_j)|)$$

И имеет реализацию:

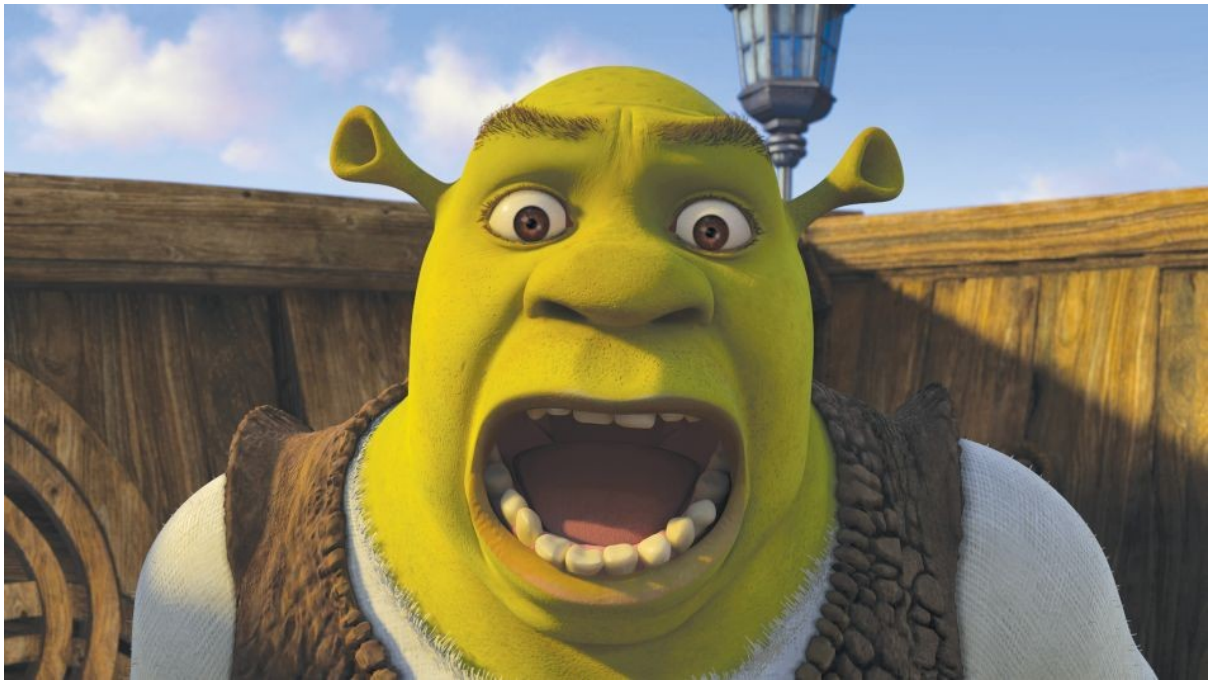
```
__device__ double D(uchar4 p, const AvgCov* feature) {
double tmp[3] = {0.};
tmp[0] = p.x - feature->avg.x;
tmp[1] = p.y - feature->avg.y;
tmp[2] = p.z - feature->avg.z;

double first[3] = {0.};
for (int i = 0; i < 3; i++) {
for (int j = 0; j < 3; j++)
first[i] += ((double)tmp[j] * feature->inverse_cov[j][i]);
}
double second = 0.;
for (int i = 0; i < 3; i++)
second += (first[i] * tmp[i]);
return (-second - log(abs(feature->det_cov)));
}
```

Номер класса с максимальной D(f) записывается в альфа канал.

Результаты

Для тестов сгенерировал классы случайным образом.

1-я фотография, разрешение 950x533.
До обработки:



После обработки , 15 случайно выбранных классов.

	5 классов	15 классов	32 класса
<<<(1,1),(1,1)>>>	2180.14	6489.18	14283.78
<<<(1,1),(1,32)>>>	95.76	258.52	509.89
<<<(1,1),(32,32)>>>	51.09	153.35	276.24
<<<(1,32),(1,32)>>>	7.17	21.2	44.95
<<<(1,32),(32,32)>>>	5.08	15.25	32.53
<<<(32,32),(32,32)>>>	3.22	9.62	20.55
<<<(128,128),(8,128)>>>	3.47	9.8	20.61
CPU	209.23	541.37	1148.56

Выводы

В рамках 3-й лабораторной работы я реализовал классификатор изображений на основе Правила Байеса. С помощью этого классификатора можно, например, обрабатывать снимки из космоса вы являть на них различные объекты. Однако, чтобы делать это корректно, нам нужны правильно подобранные классы и пиксели в них.
Относительно тестов можно отметить, что версия на устройстве начинает выигрывает в производительности начиная с 1 блока с 32 потоками. При работе ядра на 128x128 блоках и 8x128 потоках виден прирост производительности в десятки раз.