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: Б. Д. Куценко
Группа: *8О-407Б-20*
Преподаватели: А.Ю. Морозов

Москва, 2023

Условие

1. Цель работы. Научиться использовать GPU для классификации и кластеризации изображений. Использование константной памяти и одномерной сетки потоков.
2. Вариант 1. Метод максимального правдоподобия.

Программное и аппаратное обеспечение

GPU: NVIDIA GeForce GTX 1650 with Max-Q Design
Compute Capability: 7.5
Global Memory: 4095 MB
Shared Memory per Block: 48 KB
Constant Memory: 64 KB
Registers per Block: 65536
Max Blocks per Grid: 2147483647
Max Threads per Block: 1024
Multiprocessors: 16
CPU: ЦП Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
Базовая скорость: 2,60 ГГц
Сокетов: 1
Ядра: 6
Логических процессоров: 12
Виртуализация: Включено
Кэш L1: 384 КБ
Кэш L2: 1,5 МБ
Кэш L3: 12,0 МБ
SSD: SAMSUNG MZVLB1T0HALR-00000
PCIe 3.0 x4
Form Factor M.2
Capacity 1 TB
Sequential Read 3500 MB/s
Sequential Write 3000 MB/s
Random Read 580K IOPS
Random Write 500K IOPS
RAM:
Планка 1 (BANK 0):
Ёмкость (Capacity): 8,589,934,592 байт
Скорость (Speed): 2,667 МГц.
Производитель (Manufacturer): SK Hynix.
Номер модели (PartNumber): HMA81GS6CJR8N-VK.
Тип памяти (SMBIOSMemoryType): 26 (DDR4 SDRAM).
Ширина данных (DataWidth): 64 бита.

Планка 2 (BANK 2):
Ёмкость (Capacity): 8,589,934,592 байт
Скорость (Speed): 2,667 МГц.
Производитель (Manufacturer): SK Hynix.
Номер модели (PartNumber): HMA81GS6CJR8N-VK.
Тип памяти (SMBIOSMemoryType): 26 (DDR4 SDRAM).
Ширина данных (DataWidth): 64 бита.
Лабораторная работа выполнялась на ОС Windows через WSL 2.0

Метод решения

Посчитать среднее значение пикселей для каждого класса.avg_j

Посчитать матрицу ковариаций для каждого класса: cov_j

Предпочитать: $-(p - cov_j)^T * {cov_j}^{-1} * (p - avg_j)$ и $\log(|\det(cov_j)|)$

Записать это в константную память. В kernel перебрать j и найти максимум функции : $-(p - cov_j)^T * {cov_j}^{-1} * (p - avg_j) - \log(|\det(cov_j)|)$.

Описание программы

Программа выполнена в одном файле:

```
__global__ void kernel(uchar4 *img, int nc, int size) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    int offset = gridDim.x * blockDim.x;
    while(idx < size) {
        uchar4 p = img[idx];
        double3 pix;
        pix.x = p.x;
        pix.y = p.y;
        pix.z = p.z;
        double max = calc(pix, 0);
        double class_ = 0;
        for(int j = 1; j < nc; ++j) {
            double curr = calc(pix, j);
            if(curr > max) {
                max = curr;
                class_ = j;
            }
        }
        img[idx].w = class_;
        idx += offset;
    }
}
__device__  double calc(double3 p, int j) {
    double3 pix;
    pix.x = p.x - AVG[j].x;
    pix.y = p.y - AVG[j].y;
    pix.z = p.z - AVG[j].z;
```

```
    double3 tmp;
    tmp.x = pix.x * INVERSED_COV[j * 9 + 0] +
            pix.y * INVERSED_COV[j * 9 + 3] +
            pix.z * INVERSED_COV[j * 9 + 6];
    tmp.y = pix.x * INVERSED_COV[j * 9 + 1] +
            pix.y * INVERSED_COV[j * 9 + 4] +
            pix.z * INVERSED_COV[j * 9 + 7];
    tmp.z = pix.x * INVERSED_COV[j * 9 + 2] +
            pix.y * INVERSED_COV[j * 9 + 5] +
            pix.z * INVERSED_COV[j * 9 + 8];
    double res =  - (tmp.x * pix.x + tmp.y * pix.y + tmp.z * pix.z) -
LOG_DET_COV[j];
    return res;
}
```

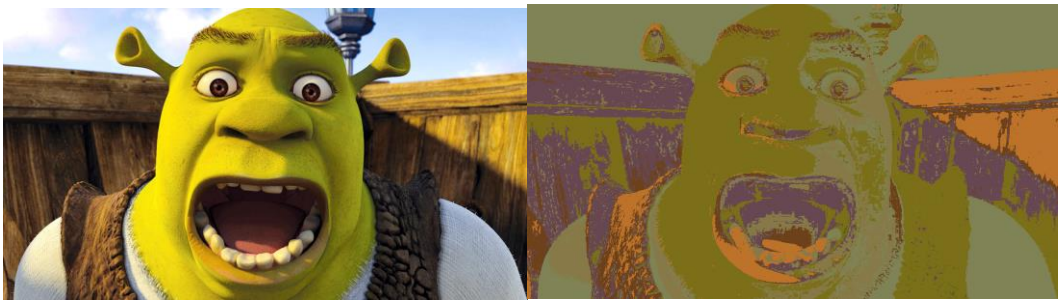
Результаты

1.

1. Grid = 1Block size = 32 time = 196.764 ms
2. Grid = 1Block size = 64 time = 74.4284 ms
3. Grid = 1Block size = 128 time = 63.6148 ms
4. Grid = 1Block size = 256 time = 63.6864 ms
5. Grid = 1Block size = 512 time = 63.5493 ms
6. Grid = 1Block size = 1024 time = 63.5465 ms
7. Grid = 2Block size = 32 time = 57.7126 ms
8. Grid = 2Block size = 64 time = 36.3332 ms
9. Grid = 2Block size = 128 time = 31.8392 ms
10. Grid = 2Block size = 256 time = 31.8304 ms
11. Grid = 2Block size = 512 time = 31.8007 ms
12. Grid = 2Block size = 1024 time = 31.7952 ms
13. Grid = 4Block size = 32 time = 28.8583 ms
14. Grid = 4Block size = 64 time = 18.1855 ms
15. Grid = 4Block size = 128 time = 15.9267 ms
16. Grid = 4Block size = 256 time = 15.9257 ms
17. Grid = 4Block size = 512 time = 15.9007 ms
18. Grid = 4Block size = 1024 time = 15.9218 ms
19. Grid = 8Block size = 32 time = 14.4376 ms
20. Grid = 8Block size = 64 time = 9.09309 ms
21. Grid = 8Block size = 128 time = 7.97011 ms
22. Grid = 8Block size = 256 time = 7.96432 ms
23. Grid = 8Block size = 512 time = 7.96262 ms
24. Grid = 8Block size = 1024 time = 7.96234 ms
25. Grid = 16Block size = 32 time = 7.22045 ms
26. Grid = 16Block size = 64 time = 4.55091 ms
27. Grid = 16Block size = 128 time = 3.98704 ms
28. Grid = 16Block size = 256 time = 3.98672 ms

29. Grid = 16Block size = 512 time = 3.98371 ms
30. Grid = 16Block size = 1024 time = 3.98989 ms
31. Grid = 32Block size = 32 time = 4.56294 ms
32. Grid = 32Block size = 64 time = 4.0855 ms
33. Grid = 32Block size = 128 time = 3.98218 ms
34. Grid = 32Block size = 256 time = 3.98304 ms
35. Grid = 32Block size = 512 time = 3.98336 ms
36. Grid = 32Block size = 1024 time = 3.98298 ms
37. Grid = 64Block size = 32 time = 3.98435 ms
38. Grid = 64Block size = 64 time = 3.98074 ms
39. Grid = 64Block size = 128 time = 3.98339 ms
40. Grid = 64Block size = 256 time = 3.98138 ms
41. Grid = 64Block size = 512 time = 3.98131 ms
42. Grid = 64Block size = 1024 time = 3.98128 ms
43. Grid = 128Block size = 32 time = 3.98096 ms
44. Grid = 128Block size = 64 time = 3.98038 ms
45. Grid = 128Block size = 128 time = 3.9816 ms
46. Grid = 128Block size = 256 time = 3.98131 ms
47. Grid = 128Block size = 512 time = 3.97978 ms
48. Grid = 128Block size = 1024 time = 3.98013 ms
49. Grid = 256Block size = 32 time = 3.97859 ms
50. Grid = 256Block size = 64 time = 3.9808 ms
51. Grid = 256Block size = 128 time = 3.97917 ms
52. Grid = 256Block size = 256 time = 3.97984 ms
53. Grid = 256Block size = 512 time = 3.97933 ms
54. Grid = 256Block size = 1024 time = 3.97773 ms
55. Grid = 512Block size = 32 time = 3.98566 ms
56. Grid = 512Block size = 64 time = 3.98141 ms
57. Grid = 512Block size = 128 time = 3.9777 ms
58. Grid = 512Block size = 256 time = 3.97786 ms
59. Grid = 512Block size = 512 time = 3.9785 ms
60. Grid = 512Block size = 1024 time = 3.97818 ms
61. Grid = 1024Block size = 32 time = 3.97722 ms
62. Grid = 1024Block size = 64 time = 3.97872 ms
63. Grid = 1024Block size = 128 time = 3.97718 ms
64. Grid = 1024Block size = 256 time = 3.97859 ms
65. Grid = 1024Block size = 512 time = 3.9791 ms
66. Grid = 1024Block size = 1024 time = 3.98784 ms
67. min is = 3.97718ms for Grid size = 1024 Block size

2. Сравнение с CPU:
   time 187 ms
3. Пример работы с изображением:



Выводы

Алгоритм может быть применен для выделения различных объектов или областей на изображениях, разделения их на классы и пометки каждого пикселя в соответствии с классом объекта.

Алгоритм может быть использован для сжатия изображений путем кодирования каждого пикселя с помощью индекса класса, к которому он относится. Это может сэкономить память и ускорить передачу изображений.