МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Факультет «Информационные технологии и прикладная математика»
Кафедра «Вычислительная математика и программирование»

Лабораторная работа №3
по курсу «Параллельная обработка данных»

Классификация

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

Москва, 2020

Условие

1. Вариант: Трех-цветовой классификатор..

Программное и аппаратное обеспечение

GPU:
Название: GeForce GTX 750TI
Графическая память: 2094071808
Разделяемая память: 49152
Константная память: 65536
Количество регистров на блок: 65536
Максимальное количество блоков: (2147483647, 65535, 65535)
Максимальное количество нитей: (1024, 1024, 64)
Количество мультипроцессоров: 5

Сведения о системе:

Процессор: AMD FX-8320 3.5Ghz
Оперативная память: 16Gb
HDD: 1Tb
Операционная система: Ubuntu 18.04
IDE: VSC
Компилятор: nvcc

Метод решения

Для каждого пикселя будем вычислять расстояние до 3 базовых цветов: (255, 0, 0), (0, 255, 0) и (0, 0, 255). Это красный, зеленый и синий. Затем из этих трех расстояний выберем минимальное. Будем говорить, что пиксель принадлежит этому классу(цвету), так как он ближе к некоторому базовому цвету.

Описание программы

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#define NAME_LEN 128
#define CUDA
#ifndef CUDA
typedef struct
{
  unsigned char x, y, z, w;
}uchar4;
#endif

__constant__ uchar3 dev_avg[3];

#define CSC(call)      \
```

```
do {                                \
   cudaError_t res = call;          \
   if (res != cudaSuccess) {        \
      fprintf(stderr, "ERROR in %s:%d. Message: %s\n",          \
            __FILE__, __LINE__, cudaGetErrorString(res));      \
      exit(0);                      \
   }                                \
} while(0)

__global__ void Classificator(uchar4* dev_image, int width, int height)
{
   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;
   uchar4 p;
   for(x = idx; x < height; x += offsetx)
   {
      for(y = idy; y < width; y += offsety)
      {
         int pos = x * width + y;
         p = dev_image[pos];
         int c1 = (p.x - dev_avg[0].x) * (p.x - dev_avg[0].x) + ((p.y - dev_avg[0].y) * (p.
y - dev_avg[0].y)) + ((p.z - dev_avg[0].z) * (p.z - dev_avg[0].z));
         int c2 = (p.x - dev_avg[1].x) * (p.x - dev_avg[1].x) + ((p.y - dev_avg[1].y) * (p.
y - dev_avg[1].y)) + ((p.z - dev_avg[1].z) * (p.z - dev_avg[1].z));
         int c3 = (p.x - dev_avg[2].x) * (p.x - dev_avg[2].x) + ((p.y - dev_avg[2].y) * (p.
y - dev_avg[2].y)) + ((p.z - dev_avg[2].z) * (p.z - dev_avg[2].z));
         c1 *= -1;
         c2 *= -1;
         c3 *= -1;

         int result = max(c1, max(c2, c3));
         if(result == c1)
            p.w = 0;
         else if(result == c2)
            p.w = 1;
         else
            p.w = 2;
         dev_image[pos] = p;
      }
   }
}
void RunOnCpu(uchar4* image, int width, int height)
{
   for(int i = 0; i < height; ++i)
```

```
    {
        for(int j = 0; j < width; ++j)
        {
            int pos = i * width + j;
            uchar4 p = image[pos];
            // printf("[%d %d] %d %d %d    ",i, j, p.x, p.y, p.z);
            int c1 = (p.x - 255) * (p.x - 255) + (p.y * p.y) + (p.z * p.z);
            int c2 = (p.x * p.x) + (p.y - 255) * (p.y - 255) + (p.z * p.z);
            int c3 = (p.x * p.x) + (p.y * p.y) + (p.z - 255) * (p.z - 255);
            c1 *= -1;
            c2 *= -1;
            c3 *= -1;

            int result = max(c1, max(c2, c3));
            // printf("c1=%d c2=%d c3=%d result = %d    ", c1, c2, c3, result);
            if(result == c1)
                p.w = 0;
            else if(result == c2)
                p.w = 1;
            else
                p.w = 2;
            image[pos] = p;
            // printf("%d\n", p.w);
        }
    }
}
int main()
{
    char input[NAME_LEN];
    char output[NAME_LEN];

    scanf("%s", input);
    scanf("%s", output);

    int width, height;
    FILE* in = fopen(input, "rb");

    fread(&width, sizeof(int), 1, in);
    fread(&height, sizeof(int), 1, in);

    uchar4* image = (uchar4*) malloc(sizeof(uchar4) * width * height);
    fread(image, sizeof(uchar4), width * height, in);
    fclose(in);

    uchar4* dev_image;
    cudaMalloc(&dev_image, sizeof(uchar4) * width * height);
    cudaMemcpy(dev_image, image, sizeof(uchar4) * width * height, cudaMemcpyHos
tToDevice);
```

```
    uchar3* avg = (uchar3*) malloc(sizeof(uchar3) * 3);
    avg[0] = make_uchar3(255, 0, 0);
    avg[1] = make_uchar3(0, 255, 0);
    avg[2] = make_uchar3(0, 0, 255);

    cudaMemcpyToSymbol(dev_avg, avg, sizeof(uchar3) * 3, 0, cudaMemcpyHostToDe
vice);

    cudaEvent_t start, end;
    CSC(cudaEventCreate(&start));
    CSC(cudaEventCreate(&end));
    CSC(cudaEventRecord(start));
    Classificator<<<dim3(2, 2), dim3(2, 2)>>>(dev_image, width, height);
    CSC(cudaEventRecord(end));
    CSC(cudaEventSynchronize(end));
    float t;
    CSC(cudaEventElapsedTime(&t, start, end));
    CSC(cudaEventDestroy(start));
    CSC(cudaEventDestroy(end));
    printf("GPU time = %.2fms\n", t);

    clock_t start_time = clock();
    RunOnCpu(image, width, height);
    printf("CPU time = %.2fms\n", (double)(clock() - start_time) * 1000 /
CLOCKS_PER_SEC);

    cudaMemcpy(image, dev_image, sizeof(uchar4) * width * height, cudaMemcpyDev
iceToHost);

    FILE* out = fopen(output, "wb");
    fwrite(&width, sizeof(int), 1, out);
    fwrite(&height, sizeof(int), 1, out);
    fwrite(image, sizeof(uchar4), width * height, out);
    fclose(out);
    free(image);
    cudaFree(dev_image);
}
```

Результаты

N	CPU(ms)	dim3(32,32) dim(32, 32)(ms)
3*3	0	0.05

N	CPU(ms)	dim3(2,2) dim(2, 2)(ms)	dim3(8,8) dim(8, 8)(ms)	dim3(16,16) dim(16, 16)(ms)	dim3(32,32) dim(32, 32)(ms)
10000*6667	1939.93	1268.78	19.11	33.9	56.42

Для такой картинки программа выведет следующие классы:

```
alex@alex-bakharew-pc:~/Documents/VII/PGP/lw3$ ./a.out < input
GPU time = 0.02ms
0 2 0
1 1 2
1 2 0
CPU time = 0.02ms
alex@alex-bakharew-pc:~/Documents/VII/PGP/lw3$
```

Выводы

Проделав данную лабораторную работу, я познакомился с алгоритмами классификации пикселей и реализовал простейший из них. В своей программе, я использовал константную память, которая доступна только для чтения для всех потоков. Это сократило использование глобальной и регистровой памяти, что при работе с большим объемом данных увеличит производительность программы.