МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

**Лабораторная работа №5-6**

**по курсу «Программирование графических процессоров»**

**Классификация и кластеризация изображений на GPU.**

Выполнил: Н.А. Зайцев

Группа: 8О-408Б

Преподаватели: К.Г. Крашенинников,

А.Ю. Морозов

Москва, 2016

**Условие**

Описание задачи:

1. **Цель работы.** Научиться использовать GPU для классификации и кластеризации изображений. Использование константной памяти.
2. **Вариант 3.** **Метод спектрального угла.**

Формат изображений соответствует формату, описанному в лабораторной работе 2. В результирующем изображении, на месте альфа­канала должен быть записан номер класса (кластера) к которому был отнесен соответствующий пиксель.

**Формат входных данных.** ​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 j­ой строки задается число ­ количество пикселей в выборке, за ним следуют пар чисел ­ координаты пикселей выборки. *nc* ≤ 32, ≤ .

- i­ый пиксель из j­ой выборки.

Оценка вектора средних:

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

**Программное и аппаратное обеспечение**

**GPU:**

Name : GeForce GTX 650

Compute capability : 3.0

Total Global Memory : 2147483648

Shared memory per block : 49152

Registers per block : 65536

Warp size : 32

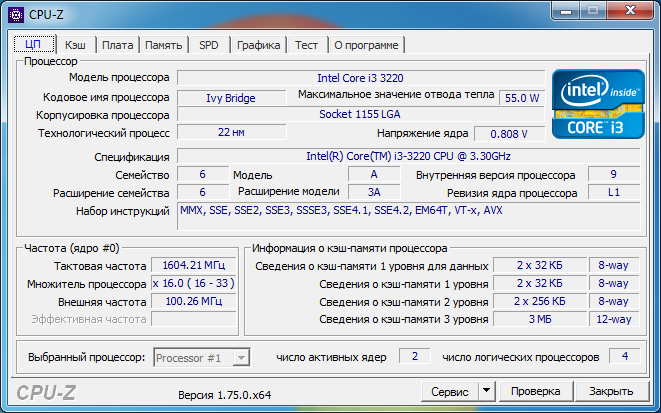
Max threads per block : (1024, 1024, 64)

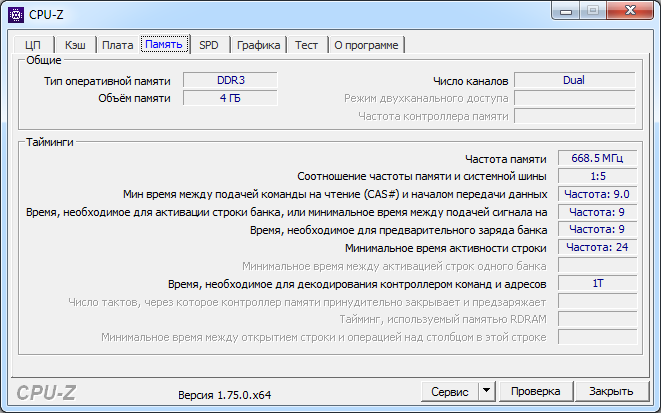
Max block : (2147483647, 65535, 65535)

Total constant memory : 65536

Multiprocessors count : 2

**CPU:**





**HDD:**

Производитель: Seagate

Модель: ST2000DM001-1CH164 ATA Device

Размер : 1,82 ТБ

**OS:**

Windows 7 Professional SP1 64-bit

**IDE:**

Microsoft Visual Studio 2013 + CUDA Plugins

**Compiler:**

nvcc

**Debugger**:

Nvidia CUDA Debugger

**Метод решения**

Программа была реализована при помощи одного ядра. Вектор средних для каждого кластера был подсчитан и отнормирован на CPU, а поиск кластера для каждого пикселя был реализован на GPU. Каждый пиксель обрабатывается однопоточно. В константой памяти находятся векторы средних.

**Описание программы**

Программа состоит из одного файла, но разбита на функцию main и одно ядро.  
  
Ядро kernel\_main:

\_\_global\_\_ void kernel\_main(int height, int width, int nc, unsigned int \*src)

{

int tid\_y = threadIdx.y + blockIdx.y \* blockDim.y;

while (tid\_y < height)

{

int tid\_x = threadIdx.x + blockIdx.x \* blockDim.x;

while (tid\_x < width)

{

double p[3];

double max = 0;

int jc = 0;

p[0] = src[tid\_y \* width + tid\_x] & 0xFF;

p[1] = (src[tid\_y \* width + tid\_x] & 0xFF00) >> 8;

p[2] = (src[tid\_y \* width + tid\_x] & 0xFF0000) >> 16;

for (int k = 0; k < nc; k++)

{

double arg = p[0] \* avg\_const[k \* 3] + p[1] \* avg\_const[k \* 3 + 1] + p[2] \* avg\_const[k \* 3 + 2];

if (arg > max)

{

max = arg;

jc = k;

}

}

src[tid\_y \* width + tid\_x] |= jc << 24;

tid\_x += blockDim.x \* gridDim.x;

}

tid\_y += blockDim.y \* gridDim.y;

}

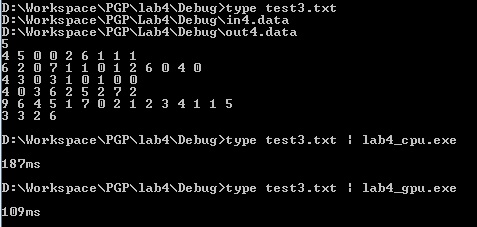
}

Используется двумерная сетка с 16-ю блоками и 16-ю потоками на каждое измерение.

Каждый поток блока обрабатывает целиком один пиксель, а затем следующий со смещением. Векторы средних находятся в константой памяти.

**Результаты**

Давайте сравним производительность программы, реализованной на CPU и GPU с использованием CUDA на тесте с изображением формата Full HD. Таймеры встроены в код программы.



Распараллеленная программа на данном тесте работает на 43,8 % быстрее. Разница ощутима.

Давайте посмотрим, как выглядит изображение на данном тесте, если в каждый цветовой канал записать номер класса, умноженный на 25.

Исходное изображение:



Обработанное изображение:



Исходное:



Обработанное:



**Выводы**

В результате выполнения данной работы я научился использовать GPU для классификации и кластеризации изображений. Научился пользоваться константной памятью.

**Использованные источники**

1. http://www.nrcgit.ru/aster/polygones/uronai/sam.htm