МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Лабораторная работа №*1*
по курсу «Технологии параллельного программирования»

Обработка изображений на GPU. Фильтры.

Выполнил: Е. С. Кострюков
Группа: М8О-407Б-22
Преподаватели: А.Ю. Морозов,
Е.Е. Заяц

Москва, 2025

Условие

Цель работы: научиться использовать GPU для обработки изображений.
Использование *текстурной памяти* и *двухмерной сетки потоков*.
Ограничение: $w < 2^{16}$ и $h < 2^{16}$.

Вариант 7. Выделение контуров. Метод Собеля.
Входные данные. На первой строке задается путь к исходному изображению, на второй, путь к конечному изображению. $w*h \leq 10^8$.

Пример:

Compute capability	hex: in.data	hex: out.data
in.data out.data	03000000 03000000 01020300 04050600 07080900 09080700 06050400 03020100 00000000 14141400 00000000	03000000 03000000 14141400 0C0C0C00 11111100 13131300 15151500 18181800 39393900 14141400 3F3F3F00
in.data out.data	03000000 03000000 00000000 00000000 00000000 00000000 80808000 00000000 00000000 00000000 00000000	03000000 03000000 B5B5B500 FFFFFF00 B5B5B500 FFFFFF00 00000000 FFFFFF00 B5B5B500 FFFFFF00 B5B5B500

Программное и аппаратное обеспечение

Аппаратная часть:

- **Процессор:** Intel(R) Xeon(R) CPU @ 2.00GHz (1 физическое, 2 логических ядра).
- **Оперативная память:** 12.67 GB.
- **Накопитель (жёсткий диск):** 73.59 GB (тип файловой системы - ext4).
- **Графический процессор (GPU):** NVIDIA Tesla T4
    - Compute Capability: 7.5.
    - Total Global Memory: 15 828 320 256 байт (~15.8 GB).
    - Shared memory per block: 49 152 байт.
    - Registers per block: 65 536.
    - Warp size: 32.
    - Max threads per block: (1024, 1024, 64).
    - Max grid size: (2147483647, 65535, 65535).
    - Total constant memory: 65 536 байт.
    - Multiprocessors count: 40.

Программная часть:

- **Операционная система:** Linux 6.6.105+ (x86_64, glibc 2.35).
- **Компилятор CUDA:** nvcc 12.5 (Cuda compilation tools, V12.5.82).
- **Интерпретатор Python:** 3.12.12.
- **Используемая IDE:** Google Colab.

Метод решения

В данной лабораторной работе рассматривается реализация алгоритма выделения контуров на изображении с использованием оператора Собеля.
Основная идея метода заключается в вычислении градиента яркости для каждого пикселя исходного изображения, что позволяет определить границы объектов.
На первом этапе из входного бинарного файла считываются размеры изображения (ширина и высота), а затем пиксели, представленные структурами типа uchar4, где хранятся компоненты цвета (R, G, B, A).
После чтения данных создаются необходимые структуры и выделяется память на стороне GPU. Изображение передаётся в двумерный массив на видеокарте, который используется в качестве текстуры. Применение текстурной памяти позволяет эффективно обращаться к пикселям с учётом граничных условий.
Затем запускается CUDA-ядро, которое в параллельном режиме обрабатывает каждый пиксель. Каждый поток вычисляет значение градиента по двум направлениям-горизонтальному и вертикальному - с использованием матриц оператора Собеля.
После этого для каждого пикселя определяется модуль градиента, и на его основе формируется новое изображение, где яркость отражает интенсивность контуров.
Результирующие данные копируются обратно в память центрального процессора и записываются в выходной бинарный файл.
В завершение программа освобождает все выделенные области памяти как на CPU, так и на GPU.

Описание программы

Программа реализована на языке C с использованием технологии CUDA и состоит из одной основной функции main, вспомогательных функций для работы с файлами и одного вычислительного ядра.
Основные функции программы:

1. **int loadImage(const char *fname, int *pw, int *ph, uchar4 **pdata)** - функция загрузки изображения из бинарного файла.
   Считывает размеры изображения, выделяет память под пиксели и заполняет массив данных.
2. **int saveImage(const char *fname, int w, int h, const uchar4 *data)** - функция сохранения результата в выходной файл в том же бинарном формате.
3. **__global__ void kernel(cudaTextureObject_t tex, uchar4 *out, int w, int h)** - вычислительное ядро CUDA, реализующее оператор Собеля.
   Каждому потоку задаются координаты пикселя (x, y). Для вычисления градиента по осям X и Y потоку требуется девять соседних значений яркости - текущий пиксель и его окружение. Яркость рассчитывается как взвешенная сумма компонент RGB по формуле
   $Y = 0.299*R + 0.587*G + 0.114*B$.
   На основе градиентов вычисляется модуль вектора, после чего результат нормируется в диапазон 0–255 и записывается в массив out.
4. **int main()** - основная функция, в которой происходит:

- считывание путей входного и выходного файлов;
- загрузка изображения с помощью loadImage;
- выделение памяти на GPU и создание текстурного объекта для исходных данных;
- запуск ядра kernel с заданными параметрами сетки и блоков;
- копирование обработанных данных обратно в память CPU;
- сохранение результата вызовом saveImage;
- освобождение всех выделенных ресурсов.

Таким образом, программа выполняет полное прохождение цикла обработки изображения: от чтения исходного файла до записи результата фильтрации. Использование текстурной памяти и параллельных вычислений позволяет значительно ускорить выполнение операций по сравнению с последовательной реализацией.

Результаты

TEST 1 (300x300)
90000
GPU timings (in ms):
grids: 1	blocks: 32	time: 3.302272 ms
grids: 1	blocks: 64	time: 1.701888 ms
grids: 1	blocks: 128	time: 0.958464 ms
grids: 1	blocks: 256	time: 0.536736 ms
grids: 1	blocks: 512	time: 0.394624 ms
grids: 8	blocks: 32	time: 0.118816 ms
grids: 8	blocks: 64	time: 0.065184 ms
grids: 8	blocks: 128	time: 0.041984 ms
grids: 8	blocks: 256	time: 0.031424 ms
grids: 8	blocks: 512	time: 0.032768 ms
grids: 64	blocks: 32	time: 0.048864 ms
|grids: 64|blocks: 64|time: 0.028672 ms|
grids: 64	blocks: 128	time: 0.028672 ms
grids: 64	blocks: 256	time: 0.034560 ms
grids: 64	blocks: 512	time: 0.052352 ms
grids: 512	blocks: 32	time: 0.751616 ms
grids: 512	blocks: 64	time: 0.645120 ms
grids: 512	blocks: 128	time: 0.514272 ms
grids: 512	blocks: 256	time: 0.782240 ms
grids: 512	blocks: 512	time: 1.683488 ms
CPU: 6.393676 ms

TEST 2 (640x640)
409600
GPU timings (in ms):

grids: 1	blocks: 32	time: 13.563264 ms
grids: 1	blocks: 64	time: 7.208896 ms
grids: 1	blocks: 128	time: 4.071520 ms
grids: 1	blocks: 256	time: 2.210016 ms
grids: 1	blocks: 512	time: 1.640448 ms
grids: 8	blocks: 32	time: 0.346432 ms
grids: 8	blocks: 64	time: 0.176608 ms
grids: 8	blocks: 128	time: 0.098560 ms
grids: 8	blocks: 256	time: 0.071968 ms
grids: 8	blocks: 512	time: 0.068672 ms
grids: 64	blocks: 32	time: 0.118784 ms
grids: 64	blocks: 64	time: 0.074272 ms
|grids: 64|blocks: 128|time: 0.061408 ms|
grids: 64	blocks: 256	time: 0.066944 ms
grids: 64	blocks: 512	time: 0.091328 ms
grids: 512	blocks: 32	time: 0.807232 ms
grids: 512	blocks: 64	time: 0.677888 ms
grids: 512	blocks: 128	time: 0.552864 ms
grids: 512	blocks: 256	time: 0.815328 ms
grids: 512	blocks: 512	time: 1.737824 ms
CPU: 28.271064 ms

TEST 3 (2048x2048)
4194304
GPU timings (in ms):
grids: 1	blocks: 32	time: 144.965027 ms
grids: 1	blocks: 64	time: 79.032318 ms
grids: 1	blocks: 128	time: 22.890495 ms
grids: 1	blocks: 256	time: 11.863424 ms
grids: 1	blocks: 512	time: 6.984096 ms
grids: 8	blocks: 32	time: 1.548288 ms
grids: 8	blocks: 64	time: 0.816032 ms
grids: 8	blocks: 128	time: 0.468960 ms
grids: 8	blocks: 256	time: 0.296960 ms
grids: 8	blocks: 512	time: 0.317472 ms
grids: 64	blocks: 32	time: 0.420576 ms
grids: 64	blocks: 64	time: 0.339968 ms
grids: 64	blocks: 128	time: 0.329312 ms
grids: 64	blocks: 256	time: 0.243168 ms
|grids: 64|blocks: 512|time: 0.237568 ms|
grids: 512	blocks: 32	time: 0.669696 ms
grids: 512	blocks: 64	time: 0.525312 ms
grids: 512	blocks: 128	time: 0.462848 ms
grids: 512	blocks: 256	time: 0.589920 ms

|grids: 512|blocks: 512|time: 1.050080 ms|
CPU: 290.511670 ms

При проведении экспериментов было выполнено сравнение времени работы программы на GPU и CPU при различных конфигурациях блоков и сетки. Замеры показали, что для малых изображений увеличение числа потоков не даёт значимого выигрыша из-за накладных расходов на запуск GPU. Для средних и больших изображений параллелизм начинает проявляться, и время работы на GPU заметно сокращается. Последовательная реализация на CPU для больших массивов существенно медленнее, тогда как GPU обеспечивает ускорение в десятки и сотни раз за счёт одновременной обработки элементов тысячами потоков. Оптимальные конфигурации блоков и сеток зависят от размера данных: слишком большое число блоков и гридов может слегка ухудшать производительность из-за увеличения нагрузки на планировщик GPU, а также конкуренции потоков.

Пример обработки изображений

Выводы

В результате данной работы было освоено программное обеспечение и методы работы с архитектурой параллельных вычислений CUDA для обработки изображений. Реализован алгоритм выделения контуров изображения с использованием фильтра Собеля на GPU, применяющий текстурную память и двумерную сетку потоков.

В процессе разработки основной сложностью стало корректное использование текстурной памяти для доступа к пикселям изображения, а также выбор оптимальных параметров блоков и сетки потоков для эффективного параллельного выполнения. Особое внимание уделялось пограничным условиям, при которых значение пикселя за границей изображения дублируется, что обеспечило корректную обработку всех областей изображения.

Проведённые эксперименты показали, что для малых изображений накладные расходы на запуск потоков GPU сопоставимы с временем вычислений, и ускорение по

сравнению с CPU невелико. Для средних изображений параллельная обработка уже эффективно использует все ресурсы GPU, и время вычислений существенно снижается. Для больших изображений GPU демонстрирует ускорение в сотни раз по сравнению с последовательной реализацией на CPU, при этом выбор конфигурации блоков и сетки потоков оказывает заметное влияние на производительность.