МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Лабораторная работа №*2*
по курсу «*Программирование графических процессоров*»
Освоение программного обеспечения для работы с технологией CUDA.

Обработка изображений на GPU. Фильтры.

Выполнил: Бачурин Павел Дмитриевич
Группа: М8О-403Б-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. Реализация одной из примитивных операций над векторами.

Вариант 7. Выделение контуров. Метод Собеля.

Входные данные:
На первой строке задается путь к исходному изображению, на второй, путь к конечному изображению. $w*h \leq 10$.

Выходные данные:
Необходимо записать в выходной файл в бинарном виде результат выполнения алгоритма Собеля над входными данными.

Программное и аппаратное обеспечение

Графический процессор (GPU):
- Модель: NVIDIA GeForce MX330
- Архитектура CUDA: Pascal
- Compute Capability: 6.1
- Видеопамять (VRAM): 2 ГБ
- Пропускная способность памяти: ~64 ГБ/с
- Количество CUDA-ядер: 384
- Разделяемая память на блок: 48 КБ
- Константная память: 64 КБ
- Количество регистров на блок: 65 536 32-битных регистров
- Максимальное количество блоков (на одном мультипроцессоре): 1024
- Максимальное количество потоков (на один блок): 1024
- Количество мультипроцессоров: 3

Центральный процессор (CPU):
- Модель: Intel i5-1035G1
- Количество ядер: 4
- Количество потоков на ядро: 2
- Тактовая частота: 3.6 ГГц

Количество оперативной памяти: 12 ГБ
Объем жесткого диска: 200 ГБ
Операционная система:
- Операционная система: Manjaro Linux Gnome x86_64
- Ядро: Linux 6.x

Программная среда: Lenovo 81WD IdeaPad 3 14IIL05
CUDA Toolkit: 12.9
Драйвер NVIDIA: 580.82.09
Компилятор gcc: 11.4.0

Метод решения

Алгоритм решения:

1) Подготовка данных на CPU:
    a) Со стандартного ввода считываются названия входного и выходного файлов
    b) Выделяется память и проверяется успешность выделения
    c) Из входного файла считывается исходное изображение
2) Копирование данных на GPU:
    a) Выделяется память на GPU (cudaMalloc).
    b) Для параллельной обработки массивы копируются в глобальную память устройства с помощью cudaMemcpy2DToArray.
    c) Создается текстурное изображение из заданных данных с помощью cudaCreateTextureObject
3) Параллельная обработка на GPU:
    a) Функция grey — вычисляет яркость (градацию серого) пикселя по стандартной формуле
    b) Функция gradient - вычисляет величину градиента яркости (интенсивность изменения цвета), используя евклидову норму и ограничивает результат максимумом 255, чтобы значение помещалось в 8-битный диапазон.
    c) Каждое GPU-ядро получает уникальные координаты (`idx`, `idy`) в сетке блоков и потоков. Эти координаты определяют, с каким участком изображения данный поток будет работать.
    d) Вычисляются смещения `offsetX` и `offsetY`, чтобы один поток мог обрабатывать несколько пикселей, если размер изображения превышает общее количество потоков.
    e) Циклы `for` по `x` и `y` с шагом `offsetX`/`offsetY` обеспечивают распределение обработки изображения между потоками без пересечений.
    f) Для каждого пикселя `(x, y)` из текстуры (`tex2D`) берутся значения 8 соседних пикселей.
    g) Каждое значение цвета преобразуется в яркость (`grey()`). Компоненты градиента `Gx` и `Gy` вычисляются с использованием разностей яркостей (по сути — аналог оператора Собеля, но в упрощённой форме). Величина градиента (контрастность на границе) вычисляется через `gradient(Gx, Gy)`. Результирующий пиксель записывается в

выходной массив `out` в градациях серого (`make_uchar4(grad, grad, grad, 0)`), где `grad` отражает силу границы.

4) Синхронизация и возврат данных:
    a) После завершения kernel все потоки синхронизируются (cudaDeviceSynchronize).
    b) Результаты копируются обратно на CPU и выводятся в выходной файл.
5) Очистка ресурсов:
    a) Освобождается память на GPU и CPU.
    b) Обрабатываются ошибки CUDA через макрос CSC(call).

Архитектура программы:

Программа построена по клиент-серверной модели памяти CPU–GPU:

1) CPU: Чтение данных, выделение памяти, контроль ошибок, вывод результатов.
2) GPU: Параллельная обработка исходного изображения через kernel. Потоки организованы в блоки и сетку, каждый поток выполняет обрабатывает свою часть изображения.
3) Взаимодействие CPU и GPU: Данные передаются на GPU и обратно через глобальную память с использованием cudaMemcpy. Время выполнения kernel измеряется с помощью событий CUDA (cudaEvent_t).

Описание программы

Основные типы данных:

1. `uchar4` — структура, содержащая четыре байта (`x`, `y`, `z`, `w`), соответствующие компонентам цвета RGBA. Используется для хранения пикселей изображения.
2. `float` — вещественный тип данных одинарной точности, применяется при вычислениях яркости и градиента.
3. `int` — целочисленный тип данных, используется для хранения размеров изображения (ширина и высота) и индексов пикселей.
4. `cudaEvent_t` — специальный тип CUDA для измерения времени выполнения вычислений на GPU.
5. `cudaArray`, `cudaTextureObject_t`, `cudaResourceDesc`, `cudaTextureDesc` — типы CUDA, используемые для создания, настройки и доступа к текстуре изображения в памяти GPU.

Основные функции и макросы:

1. Макрос `CSC(call)`
   Используется для проверки ошибок при вызовах CUDA API.

Если вызов возвращает ошибку, выводится сообщение с указанием файла, строки и текста ошибки, после чего программа аварийно завершается.

2. Функция `grey(uchar4 p)`
   Устройство (`__device__`) функция, вычисляющая яркость пикселя.

   Возвращает значение яркости в диапазоне от 0 до 255.

3. Функция `gradient(float Gx, float Gy)`
   Устройство (`__device__`) функция, вычисляющая величину градиента по направлениям X и Y

Результаты

1. Зависимость времени выполнения программы от количества используемых потоков (для тестов использовались изображения весом 3мб):

Потоки	Время (в мс)
8 × 8, 8 × 8	30
64 × 64, 64 × 64	10
128 × 128, 128 × 128	6

2. Сравнение программы на CUDA и программы на CPU с одним потоком:

Размер изображения	Время на CUDA (в мс)	Время на CPU (в мс)
3 мб	6	1203

Выводы

В ходе выполнения лабораторной работы была реализована и протестирована программа для обработки изображений на графическом процессоре (GPU) с использованием технологии CUDA. В рамках лабораторной работы была реализована фильтрация изображения методом Собеля, позволяющая выделять контуры и границы объектов. Для хранения исходных данных использовалась текстурная память, обеспечивающая эффективный доступ к пикселям, а вычисления выполнялись параллельно в нескольких потоках CUDA. Полученные результаты подтвердили эффективность использования GPU для задач обработки изображений — вычисления выполняются значительно быстрее по сравнению с последовательной реализацией на CPU.
Таким образом, в ходе работы были закреплены навыки работы с памятью GPU, текстурами, сеткой потоков и синхронизацией CUDA-ядер.