МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №0
по курсу «Программирование графических процессоров»

Освоение программного обеспечения для работы с технологией CUDA. Примитивные операции над векторами. Вариант 4. Поэлементное нахождение минимума векторов.

Выполнил: А.В. Синявский
Группа: 8О-408Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2021

Условие

Цель работы.

Ознакомление и установка программного обеспечения для
работы с программно-аппаратной архитектурой параллельных вычислений(CUDA).
Реализация одной из примитивных операций над векторами.

Вариант 4. Поэлементное нахождение минимума векторов.

Входные данные. На первой строке задано число n -- размер векторов. В следующих 2-х строках, записано по n вещественных чисел -- элементы векторов.

Выходные данные. Необходимо вывести n чисел -- результат поэлементного нахождения минимума исходных векторов.

Пример:

Входной файл	Выходной файл
3 1 5 3 4 2 6	1.0000000000e+00 2.0000000000e+00 3.0000000000e+00

Программное и аппаратное обеспечение

Nvidia GeForce GTX 660

Compute capability: 3.0
Графическая память: 2048MB
Регистров на блок: 65536
Нитей на блок: 1024
Мультипроцессоров: 5
Всего ядер: 960

Intel(R) Core(TM) i5-3570 CPU @ 3.40GHz

Тактовая частота: 3.4 GHz
Кэш-память: 6 MB

Оперативная память

Объём: 8 GB

Жёсткий диск

Объём: 2 TB

Программное обеспечение

OS: Windows 10
IDE: Visual Studio 2019
CUDA: v10.2 nvcc

Метод решения

Алгоритм предельно прост: загружаем оба вектора в память устройства, на нём сравниваем элементы, по мере необходимости перезаписываем значение элементов первого вектора. Первый вектор копируем обратно в оперативную память, выводим. Не забываем ловить возможные ошибки на GPU и чистить за собой память.

Описание программы

1. Макрос CSC
   Проверяет, с каким статусом завершаются CUDA-операции, и в, случае ошибки, выводит на стандартный поток ошибок debug-информацию.

2. Ядро.
   Основная функция, работающая на устройстве. Принимает указатели на вектора (лежащие на GPU), сравнивает их значения, перезаписывает значение элемента первого вектора, если соответствующий элемент второго вектора меньше.

3. Main
   Тут реализован ввод входных данных, вывод результата, выделение памяти, копирование векторов на устройство, и вызов функции-ядра.

Результаты

Размер теста	<<<1,32>>>	<<<32,32>>>	<<<128,128>>>	<<<256,256>>>	<<<1024,1024>>>	CPU
100	0.03	0.014	0.014	0.0175	0.15	1e-06
1000	0.18	0.013	0.014	0.016	0.15	7e-06
10000	1.9	0.34	0.295	0.3	0.15	7.1e-05
100000	17.5	0.8	0.35	0.35	0.47	0.0007
1000000	170.62	5.67	0.96	0.87	0.97	0.007
10000000	1561.32	54.07	7.1	6.29	6.03	0.0754

Выводы

Данная работа была составлена исключительно в обучающих целях, поэтому применение алгоритму придумать несколько затруднительно. Возможно, поэлементное сравнение векторов с выбором минимума можно использовать для наложения достаточно сложного неоднородного фильтра на изображение: у нас есть «маска» (матрица с числами), каждый элемент которой отражает, до какого значения нужно понизить один из RGB-параметров соответствующего пикселя в изображении. «Маска» поэлементно сравнивается с изображением, выбирается минимум, красного цвета в определённых маской областях становится меньше.

Написать код решения было нетрудно, так как задание практически идентично примеру, который студентам показали на лекции. Основные сложности были связаны с установкой CUDA для работы с моей не самой новой видеокартой, а также с работой в Visual Studio. В данном IDE я работал впервые, и разобраться во многих функциях было довольно затруднительно.

Из таблицы замера времени можно сделать ряд наблюдений. Во-первых, алгоритм на CPU работает за линейную от размера тестов сложность. Во-вторых, чем больше блоков и нитей в них мы выделяем для ядра, тем бОльшие по размеру тесты алгоритм обрабатывает с сублинейной сложностью. Например, при кофигурации 1,32 уже между вторым и третьим тестами видна линейная зависимость, а при конфигурации 1024,1024 чистой линейной зависимости не прослеживается ни в одном месте в цепочке тестов. Из этих наблюдений можно сделать вывод о высокой эффективности распараллеливания задач на графическом процессоре.