МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

**Лабораторная работа №*1***

**по курсу «Программирование графических процессоров»**

***Освоение программного обеспечения для работы с технологией CUDA.***

***Примитивные операции над векторами***

Выполнил: Р.С. Лисин

Группа: 8О-406Б

Преподаватели: К.Г. Крашенинников,

А.Ю. Морозов

Москва, 2023

**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.

**Программное и аппаратное обеспечение**

В качестве графического процессора использую видеокарту NVIDIA Tesla T4.

Compute capability : 7.5

Name : Tesla T4

Total Global Memory : 15835398144

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 : 40

В качестве редактора кода использовался Jupyter Notebook в Google Colab.

**Метод решения**

Поэлементное нахождение минимума векторов - это операция, которую можно легко распараллелить, потому что она выполняется поэлементно. Поэтому на графическом процессоре будем находить минимум поэлементно у двух массивов.

**Описание программы**

В программе создаются два динамических массива arr1, arr2. Они копируются на GPU. В функции ядра kernel происходит поэлементное нахождение минимума элементов массивов. Результат записывается в первый массив, чтобы не использовать лишнюю память.

**Результаты**

Рассмотрим время работы программы на различных тестах при различных размерах сетки и на CPU. Будем замерять непосредственно время работы алгоритма. В качестве тестов используются векторы из дробных элементов различной длины. Результаты приведены в таблице ниже.

|  |  |  |  |
| --- | --- | --- | --- |
| Размер сетки ядра | 1000 элементов, мс | 10000 элементов, мс | 100000 элементов, мс |
| CPU | 0.012000 | 0.094000 | 1.643000 |
| <<<1, 32>>> | 0.059776 | 0.204096 | 1.613824 |
| <<<32, 32>>> | 0.053216 | 0.035392 | 0.067808 |
| <<<128, 128>>> | 0.028768 | 0.036384 | 0.025568 |
| <<<256, 256>>> | 0.035936 | 0.028224 | 0.027072 |
| <<<512, 512>>> | 0.028704 | 0.033664 | 0.036528 |
| <<<1024, 1024>>> | 0.037056 | 0.042784 | 0.046176 |

На небольших данных алгоритм на CPU справляется быстрее параллельного. Но чем больше данных в тесте, тем сильнее заметна разница между временем выполнения алгоритма на CPU и на графическом процессоре.

**Выводы**

В первой лабораторной работе я познакомился с технологией CUDA, попробовал реализовать и запустить простой алгоритм с использованием вычислений на графическом процессоре, а также сравнил время работы алгоритма при реализации на CPU и при реализации на различных конфигурациях графического процессора. Заметил, что на больших объемах данных алгоритм на GPU работает до 3 раз быстрее! Поэтому видеокарты все чаще используют для выполнения объёмных вычислительных задач.