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: *А. Ю. Голов*

Группа: *М8О-401*

Преподаватель: А.Ю. Морозов

Москва, 2025

**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 3. Поэлементное умножение векторов.

**Программное и аппаратное обеспечение**

Видеокарта: NVIDIA GeForce RTX 2060 Mobile

* **Compute Capability**: 7.5
* **Графическая память**: 6 ГБ GDDR6, с 192-битной шиной и пропускной способностью 336 ГБ/с
* **Разделяемая память**: до 64 КБ на мультипроцессор
* **Константная память**: 64 КБ
* **Количество регистров на блок**: 65 536
* **Максимальное количество блоков на мультипроцессор**: 16
* **Максимальное количество нитей на мультипроцессор**: 1 024
* **Количество мультипроцессоров (SM)**: 30

Процессор: **Intel Core i7-9750H** имеет следующие характеристики:

* **Количество ядер**: 6
* **Количество потоков**: 12
* **Техпроцесс**: 14 нм

Оперативная память:

* Объём: 16 ГБ
* **Тактовая частота:** 3500 MHz
* **Поколение**: DDR4

Жёсткий диск:

* **Объём: 512 ГБ**
* Формат: SSD M2

Программное обеспечение:

* **Операционная система: Ubuntu 24.04 LTS**
* IDE: Lunar Vim

**Метод решения**

### **Методика решения задачи поэлементного умножения векторов с использованием CUDA**

1. **Определение параметров CUDA**
   * **Количество потоков на блок (**THREADS\_AMOUNT**)** и **количество блоков (**BLOCKS\_AMOUNT**)** задаются заранее.
   * Определяется общее количество потоков (shift), которые будут обрабатывать элементы векторов.
2. **Выделение памяти на устройстве (GPU)**
   * Для хранения входных векторов (a и b) и результирующего вектора (res) выделяется память с помощью cudaMalloc.
3. **Копирование данных с хоста на устройство**
   * Входные векторы копируются в видеопамять (cudaMemcpy).
4. **Запуск ядра CUDA (**kernel**)**
   * Каждому потоку присваивается свой индекс (i).
   * Внутри ядра выполняется **поэлементное умножение** res[i] = a[i] \* b[i].
   * Если число потоков меньше размера вектора, используется **шагающий доступ** (i += shift), чтобы все элементы были обработаны.
5. **Копирование результатов обратно на хост**
   * После завершения работы ядра результаты копируются обратно из памяти GPU в память CPU (cudaMemcpy).
6. **Вывод результата**
   * Значения результирующего вектора выводятся с фиксированной точностью (std::fixed << std::setprecision(10)).
7. **Очистка памяти**
   * Освобождается выделенная на GPU память (cudaFree).

**Описание программы**

**Реализованное ядро:**

\_\_global\_\_ void kernel(double \*a, double \*b, double \*res, int size)

{

int shift = gridDim.x \* blockDim.x;

for (int i = blockIdx.x \* blockDim.x + threadIdx.x; i < size; i += shift) {

res[i] = Multiply(a[i], b[i]);

}

}

1. **Определение** shift
   * gridDim.x \* blockDim.x вычисляет **общее количество потоков в сетке**, чтобы учесть все возможные потоки.
2. **Вычисление индекса** i
   * blockIdx.x \* blockDim.x + threadIdx.x задаёт **уникальный индекс потока** в массиве.
3. **Шагающий доступ (**i += shift**)**
   * Позволяет **одному потоку обработать несколько элементов**, если потоков меньше, чем размер массива.
4. **Поэлементное умножение**
   * Каждый поток выполняет res[i] = Multiply(a[i], b[i]);, вычисляя произведение соответствующих элементов.

**Результаты**

Время исполнения программного кода указывается в секундах

|  |  |  |  |
| --- | --- | --- | --- |
| n | Конфигурация | GPU | CPU |
| 100 | <<<1, 32>>> | 0.000210211 | 3.689e-06 |
| <<<16, 64>>> | 2.6451e-05 |
| <<<64, 256>>> | 6.69e-06 |
| <<<512, 512>>> | 2.7169e-05 |
| <<<1024, 1024>>> | 3.018e-05 |
| 100000 | <<<1, 32>>> | 0.0015234120 | 0.000620464 |
| <<<16, 64>>> | 0.0016414780 |
| <<<64, 256>>> | 0.0000114940 |
| <<<512, 512>>> | 0.0000111080 |
| <<<1024, 1024>>> | 0.0000197380 |
| 10000000 | <<<1, 32>>> | 0.1531488900 | 0.0803595 |
| <<<16, 64>>> | 0.0072167550 |
| <<<64, 256>>> | 0.0013055320 |
| <<<512, 512>>> | 0.0008313300 |
| <<<1024, 1024>>> | 0.0008271660 |

**Выводы**

Алгоритм предназначен для **поэлементного умножения векторов** на **GPU**. Используется в **обработке данных, графике, машинном обучении и моделировании физических процессов**.

Программирование требует знаний **CUDA, потоков, блоков и работы с памятью**. Основные сложности: **балансировка нагрузки, передача данных между CPU и GPU, оверхед синхронизации**.

### **Сравнение результатов**

* При **малых данных (**n=100**)** затраты на передачу превышают выгоду от GPU.
* **На больших массивах (**n=10^7**)** ускорение заметно.
* **Оптимальное соотношение блоков и потоков** (<<<64, 256>>>, <<<512, 512>>>) даёт лучшие результаты.

Для **больших данных** GPU превосходит CPU, но важно **оптимально подбирать количество потоков и блоков**.