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: О.А. Мезенин
Группа: 8О-406Б
Преподаватель: А.Ю. Морозов

Москва, 2024

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 2: Вычитание 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 вычитания исходных векторов.

Программное и аппаратное обеспечение

Характеристики графического процессора:

- Compute capability: 6.1
- Name: NVIDIA GeForce GTX 1060 6GB
- Total Global Memory: 6352011264
- 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: 10

Характеристики центрального процессора:

- Version: AMD Ryzen 5 2600 Six-Core Processor
- External Clock: 100 MHz
- Max Speed: 3900 MHz
- Current Speed: 3400 MHz
- Core Count: 6
- Thread Count: 12

Характеристики оперативной памяти:

- Size: 2x8 GB
- Type: DDR4
- Speed: 2133 MT/s

Характеристики жесткого диска:

- Device Model: ADATA SU635
- User Capacity: 480 103 981 056 bytes [480 GB]
- Sector Size: 512 bytes logical/physical
- SATA Version is: SATA 3.2, 6.0 Gb/s (current: 6.0 Gb/s)

Программное обеспечение:

- OS: Linux version 6.9.12-3-MANJARO
- Текстовый редактор: Kate
- Компилятор: nvcc V12.4.131

Метод решения

Для вычисления разности векторов пройдемся индексом от 0 до n, и сделаем поэлементное вычитание, сохраняя разность в результирующий вектор: res[i] = v1[i] – v2[i].

Описание программы

Вся программа описана в одном файле lab.cu. Есть следующие функции:

- int main() — входная точка программы, которая считывает массивы, запускает вычисления и выводит результат;
- double *readVector(int n) — выделяет массив размера n и считывает числа из ввода;
- void printVector(double *v, int n) — выводит массив v размера n;
- __global__ void kernel(double *v1, double *v2, double *res, int n) — ядро вычисляет разность массивов v1 и v2 размера n и кладёт результат в массив res.

Результаты

Тесты будут следующих размеров: n=10, n=10000, n=10000000.
Числа в тестах с плавающей запятой и варьируются от -1000 до 1000.
Далее приведены замеры времени работы ядер с тремя конфигурациями, а также замеры работы без использований технологий CUDA (на CPU):

1. Конфигурация <<<1, 32>>>

n	Время (в мс)
10	0.133120
10000	0.200960
10000000	105.314140

2. Конфигурация <<<64, 32>>>

n	Время (в мс)
10	0.130048
10000	0.134912
10000000	2.516128

3. Конфигурация <<<1024, 1024>>>

n	Время (в мс)
10	0.141312
10000	0.143488
10000000	1.733696

4. CPU

n	Время (в мс)
10	0
10000	0.06
10000000	48.658

Выводы

В ходе лабораторной работы ознакомился с программным обеспечением для работы с CUDA, а также реализовал алгоритм вычитания векторов и провёл сравнительный

анализ работы алгоритма с использованием технологий CUDA и без них. Столкнулся с проблемой несоответствия версии CUDA Toolkit и версией драйвера, из-за чего не компилировалась программа, но эта проблема исправилась путём установки более старой версии CUDA Toolkit. Из результатов сравнения можно увидеть, что конфигурация <<<1024, 1024>>> справляется лучше всех с большим тестом, при этом с большим отрывом от конфигурации <<<1, 32>>>. Интересно, что выполнение алгоритма на CPU работает быстрее на маленьком и среднем тестах, однако заметно медленнее работает на большом тесте, если сравнивать с конфигурациями <<<64, 32>>> и <<<1024, 1024>>>. Из этого можно сделать вывод, что при правильной настройке конфигурации, т.е. выборе блоков и потоков, можно получить значительный прирост производительности на больших входных данных.