МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии 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: М.А. Жерлыгин
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 №3: Поэлементное умножение векторов.

Программное и аппаратное обеспечение

GPU:

1. Compute capability: 7.5;
2. Графическая память: 4294967296;
3. Разделяемая память: 49152;
4. Константная память: 65536;
5. Количество регистров на блок: 65536;
6. Максимальное количество блоков: (2147483647, 65535, 65535);
7. Максимальное количество нитей: (1024, 1024, 64);
8. Количество мультипроцессоров: 6.

Сведения о системе:

1. Процессор: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz
2. Память: 16,0 ГБ;
3. HDD: 237 ГБ.

Программное обеспечение:

1. OS: Windows 10;
2. IDE: Visual Studio 2019;
3. Компилятор: nvcc.

Метод решения

Для нахождения поэлементного умножения векторов нужно в цикле умножить каждый элемент вектора X на соответствующий ему элемент вектора Y.

Описание программы

Для начала считываем размер вектора и значения обоих векторов. После, с помощью cudaMalloc() и cudaMemcpy() выделяем память и копируем данные в выделенный массив и вызываем kernel.

В самом kernel мы выполняем поэлементное умножение векторов:

```
__global__ void kernel(double *arr1, double *arr2, int size) {
    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    int offset = blockDim.x * gridDim.x;
    while (idx < size) {
        arr1[idx] = arr1[idx] * arr2[idx];
        idx += offset;
    }
}
```

После копируем из выделенного массива данные обратно и выводим.

Результаты

CPU:

Количество	10^5	10^6	10^7	$3 * 10^7$	$6 * 10^7$	$9 * 10^7$	10^8
Time	< 0.001	0.001	0.055	0.161	0.302	0.460	0.584

Ядро<<1, 32>>

Количество	10^5	10^6	10^7	$3 * 10^7$	$6 * 10^7$	$9 * 10^7$	10^8
Time	0.003711	0.037281	0.310421	0.937484	1.803214	2.70731	3.020543

Ядро<<32, 32>>

Количество	10^5	10^6	10^7	$3 * 10^7$	$6 * 10^7$	$9 * 10^7$	10^8
Time	0.00012	0.001221	0.0122	0.035523	0.056475	0.087318	0.096945

Ядро<<64, 64>>

Количество	10^5	10^6	10^7	$3 * 10^7$	$6 * 10^7$	$9 * 10^7$	10^8
Time	0.00004	0.000349	0.003491	0.01018	0.017312	0.026164	0.028927

Ядро<<128, 128>>

Количество	10^5	10^6	10^7	$3 * 10^7$	$6 * 10^7$	$9 * 10^7$	10^8
Time	0.000025	0.000221	0.002149	0.006391	0.013236	0.019658	0.021895

Выводы

Я ознакомился с технологией CUDA и применил её на примере простой задачи поэлементного умножения двух векторов. На основе полученных измерений времени выполнения работы программы, можно сделать вывод, что при некоторых конфигурациях ядра, на CPU задача выполняется быстрее. Однако, при грамотном подборе конфигурации ядра, можно достигнуть улучшения скорости работы программы по сравнению с CPU (особенно заметно на больших объемах данных).