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: Д. А. Ваньков
Группа: 8О-407Б-17
Преподаватели: А. Ю. Морозов,
К. Г. Крашенинников

Москва, 2020

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 7. Поэлементное вычисление модуля вектора.

Программное и аппаратное обеспечение

Graphics card: GeForce 940M
Размер глобальной памяти: 4242604032
Размер константной памяти: 65536
Размер разделяемой памяти: 49152
Максимальное количество регистров на блок: 65536
Максимальное количество потоков на блок: 1024
Количество мультипроцессоров: 3
OS: Linux Ubuntu 18.04
Редактор: CLion, Atom

Метод решения

Для нахождения поэлементного модуля вектора нужно в цикле проверить является ли число отрицательным, и если является изменить его по модулю.

Описание программы

Для того, чтобы выполнить поэлементную операцию взятия числа по модулю необходимо создать вектор длинны n на device и заполнить его элементами. Затем, я скопировал данные из вектора в выделенный массив с помощью функции cudaMemcpy и передал их в kernel. После работы kernel я скопировал результат в выходной вектор с помощью аналогичной функции.

Для запуска kernel на device необходимо задать количество блоков и потоков в каждом из блоков. Для одномерного массива нам достаточно вызывать блоки и нити в одном измерении. Вызов kernel с количеством нитей на блок — 256 и количеством блоков, достаточным для того, чтобы каждому отдельному потоку достался один элемент из вектора:

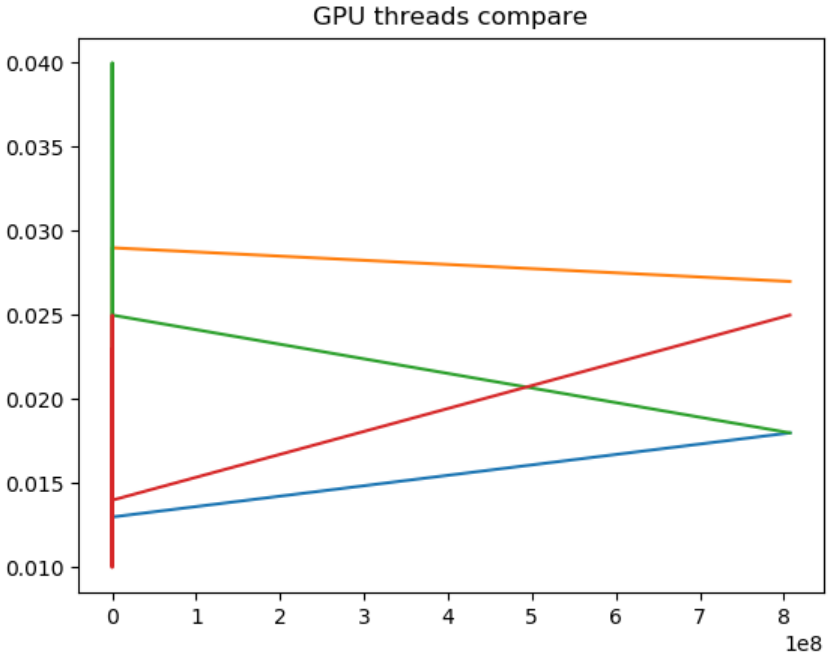
```
kernel<<<256, 256>>>(dev_arr, n);
```

В самом kernel я вычисляю общий индекс исполняемой нити который и будет индексом в массиве при условии idx < array.size(). Далее выполняю операцию нахождения модуля элемента вектора и перезаписи значения в текущий индекс:

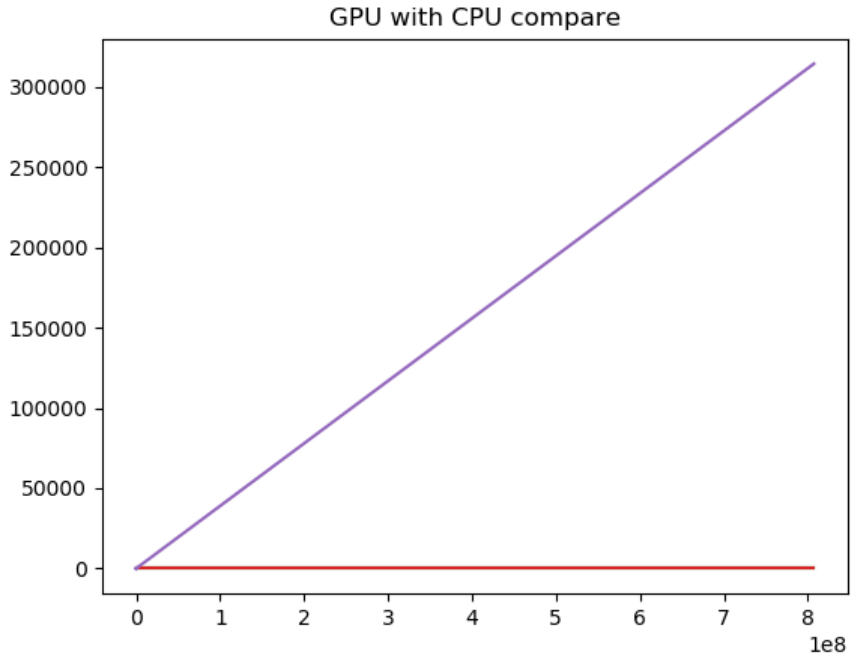
```
__global__ void kernel(float *arr, int n) {
	int idx = blockDim.x * blockIdx.x + threadIdx.x;	// Абсолютный номер потока
	int offset = blockDim.x * gridDim.x;			// Общее кол-во потоков
	for(int i = idx; i < n; i += offset) {
		if (arr[i] < 0)
			arr[i] = abs(arr[i]);
	}
}
```

Результаты

После тестирования программы на различных данных и с отличающимся количеством блоков и нитей были получены следующие результаты. Из графика можно заметить, что лучше всего оказался запуск с 32 блоками, по 32 нити в каждом. Скорее всего, это происходит из-за скорости переключения между нитями, которая выше, чем другого количества нитей внутри блоков. По оси OX - размер теста, OY - время исполнения в ms.



Также, я провел сравнение времени непосредственных вычислений между CPU и GPU, не беря в учет время копирования векторов с hosta на gpu. Из графика можно заметить, что разница колоссальная. По оси OX - размер теста, OY - время исполнения в ms.



Пример вывода логов:

CPU:	GPU Threads: 32, 32:
n: 807268796	n: 807268796
314514ms	0.018ms

Выводы

Данная лабораторная не вызвала у меня трудностей, так как это вводное задание в курс программирования графических процессоров.

Можно также заметить, что способ вычисления значений на CPU, при небольших данных не отличается от GPU, а иногда даже является более быстрым, таким образом GPU целесообразнее использовать на больших данных, где преимущество становится заметным.