Министерство цифрового развития, связи
и массовых коммуникаций Российской Федерации

Сибирский государственный университет
телекоммуникаций и информатики

Кафедра прикладной математики и кибернетики

ЛАБОРАТОРНАЯ РАБОТА №3

По дисциплине: «Программирование графических процессоров»

Выполнили:

Студенты 3 курса группы ИП-111
Корнилов А.А.,
Попов М.И.,
Толкач А.А.

Проверил:

Профессор кафедры ПМиК
Малков Е.А.

Новосибирск, 2024

Задание: 1. Напишите программу сложения двух векторов на GPU.

2.Зафиксируйте длину векторов равной 1<<20.

3. Проведите серию запусков программы варьируя количество нитей в блоке, 1, 16, 32, 64,128, …, 1024, для запуска ядра.

3. Определите зависимость времени выполнения ядра, вычисляющего сумму векторов, от конфигурации нитей. Используйте для определения времени события CUDA и профилировщики.

Цель: априорное понимание влияния конфигурации нитей на производительность выполнения кода на GPU.

Оборудование: GeForce GTX1050ti

Выполнение работы:

Для первого задания была написана простоя программа для сложения двух векторов используя GPU, количество векторов задано n = 1<<20, для определения времени работы используется CudaEvent

```
#include <iostream>
#include <cstdlib>
#include <cuda.h>

#include "device_launch_parameters.h"

using namespace std;

const int n = 1 << 20;

__global__ void vectorAdd(int arr1[], int arr2[]) {
    unsigned int i = threadIdx.x + blockDim.x * blockIdx.x;
    arr1[i] += arr2[i];
}

int main() {
    int *local_arr1, *local_arr2, *GPU_arr1, *GPU_arr2;
    int threads_per_block, num_of_blocks;
    float time;

    cout << "Enter threads num: ";
    cin >> threads_per_block;
    num_of_blocks = n / threads_per_block;

    cudaEvent_t start, stop;
```

```
    cudaEventCreate(&start);
    cudaEventCreate(&stop);

    cudaMalloc((void **) &GPU_arr1, n * sizeof(int));
    cudaMalloc((void **) &GPU_arr2, n * sizeof(int));
    local_arr1 = (int *) calloc(n, sizeof(int));
    local_arr2 = (int *) calloc(n, sizeof(int));


    for (int i = 0; i < n; i++) {
        local_arr1[i] = i;
        local_arr2[i] = i + 1;
    }

    cudaMemcpy(GPU_arr1, local_arr1, n * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(GPU_arr2, local_arr2, n * sizeof(int), cudaMemcpyHostToDevice);

    cudaEventRecord(start, nullptr);
    vectorAdd<<<dim3(num_of_blocks),dim3(threads_per_block)>>>(GPU_arr1, GPU_arr2);
    cudaEventRecord(stop, nullptr);
    cudaEventSynchronize(stop);

    cudaEventElapsedTime(&time, start, stop);

    cout << "CUDA Event time:\n\t"
         << time <<"ns"
         << endl;

    cudaFree(GPU_arr1);
    cudaFree(GPU_arr2);
    delete[] local_arr1;
    delete[] local_arr2;

    return 0;
}
```

Листинг 1 – программа LR03_1.cu

Команда компиляции и результат работы программы, количество выбранных нитей выбирается с клавиатуры:

```
Enter threads num:16
 16
CUDA Event time:
      0.441408ns


Process finished with exit code 0
```

Ниже приведена таблица (таб. 1) с запуском программы с количеством нитей от 1 до 1024. Также графики с использование CudaEvent (грф. 1) и таймера Chrono (грф. 2). По графикам можно сделать вывод что время выполнения уменьшается относительно увеличения количества нитей

Нити	EventTime
1	5,47811
2	3,76611
4	1,44253
8	0,758624
16	0,453152
32	0,242528
64	0,217376
128	0,167424
256	0,167776
512	0,170976
1024	0,171296

Таблица 1 – Запуск программы с различным количеством нитей

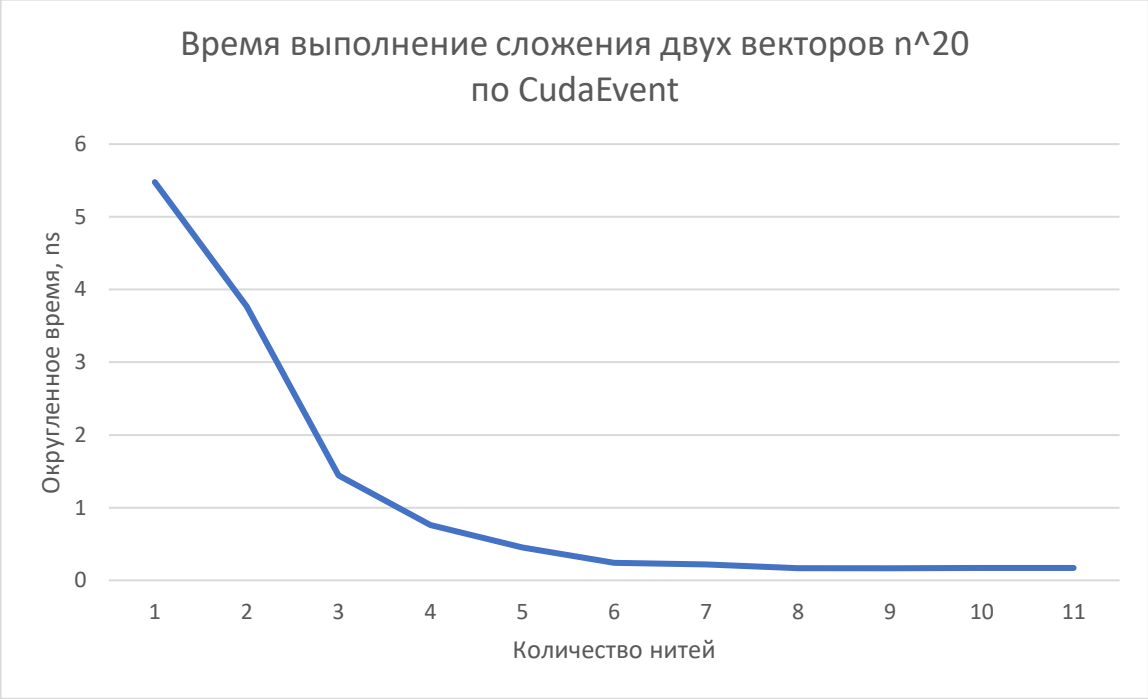

График 1 – время выполнения от кол-во нитей по таймеру CudaEvent

Вывод: Мы вычислили что для GTX 1050ti будет оптимально использовать 128 нитей на блок