Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Сибирский государственный университет
телекоммуникаций и информатики» (СибГУТИ)

09.03.01 "Информатика и вычислительная техника"
профиль "Программное обеспечение средств
вычислительной техники и автоматизированных систем"

ОТЧЕТ
по дисциплине «Программирование графических процессоров»
лабораторная работа 2

Выполнил:
Разумов Д. Б.
студент группы ИП-811

Проверил:
преподаватель Малков Е.А.

Новосибирск 2020

Оглавление

Задание

Задание: выполнить задание лабораторной 1, используя Events и nvprof.

Цель: научиться обрабатывать ошибки и профилировать код при выполнении программы на GPU.

Листинг

```
#include <stdio.h>
#include <time.h>
#include <malloc.h>

#define CUDA_CHECK_RETURN(value) {\
	cudaError_t _m_cudaStat = value;\
	if (_m_cudaStat != cudaSuccess) {\
		fprintf(stderr, "Error \"%s\" at line %d in file %s\n",\
			cudaGetErrorString(_m_cudaStat), __LINE__, __FILE__);\
		exit(1);\
	}\
} //макрос для обработки ошибок

const int N = 1 << 20;

__global__ void gInitVectors(double* vector1, double* vector2) {
	for (int i = 0; i < N; i++) {
		vector1[i] = (double)i; //rand();
		vector2[i] = (double)i;
	}
}

__global__ void gVectorAddition(double* vector1, double* vector2, double*
vectorSum, int threads_cnt) {
	int i = threadIdx.x + blockIdx.x * blockDim.x;
	if (i >= N)
```

```
        return;
    vectorSum[i] = vector1[i] + vector2[i];

}

float testingThreadsOfDevice(int threads_cnt) {
    double *vectorSum_d, *vectorSum_h;
    vectorSum_h = (double*) calloc(N, sizeof(double));
    cudaMalloc((void**)&vectorSum_d, N * sizeof(double));
    double *vector1_d, *vector2_d;
    cudaMalloc((void**)&vector1_d, N * sizeof(double));
    cudaMalloc((void**)&vector2_d, N * sizeof(double));
    gInitVectors <<< 1, 1 >>> (vector1_d, vector2_d);

    /*
    struct timespec mt1, mt2;
    clock_gettime(CLOCK_REALTIME, &mt1);
    */
    float elapsedTime;
    cudaEvent_t start, stop; // встроенный тип данных – структура, для
    // фиксации контрольных точек
    cudaEventCreate(&start); // инициализация
    cudaEventCreate(&stop); // событий
    cudaEventRecord(start,0); // привязка (регистрация) события start

    gVectorAddition <<< N / threads_cnt, threads_cnt >>>
        (vector1_d, vector2_d, vectorSum_d, threads_cnt); //запуск фу-ии на
GPU
```

```
	cudaDeviceSynchronize(); //синхронизация потоков


	cudaEventRecord(stop,0); // привязка события stop
	cudaEventSynchronize(stop); // синхронизация по событию
	//CUDA_CHECK_RETURN(cudaDeviceSynchronize());
	CUDA_CHECK_RETURN(cudaGetLastError());
	cudaEventElapsedTime(&elapsedTime,start,stop); // вычисление затраченного времени
	cudaEventDestroy(start); // освобождение
	cudaEventDestroy(stop); // памяти


	/*
	clock_gettime(CLOCK_REALTIME, &mt2);
	double seconds_double = (double)(mt2.tv_sec - mt1.tv_sec) +
	(double)(mt2.tv_nsec - mt1.tv_nsec) / 1e9; ///время в секундах
	*/
	printf("blocks = %d, threads per block = %d seconds = %e \n",
		N / threads_cnt, threads_cnt, elapsedTime);



	/// проверка: ///
	/*cudaMemcpy(vectorSum_h, vectorSum_d, N * sizeof(double), cudaMemcpyDeviceToHost);
	for (int i = 0; i < N; i++)
		fprintf(stderr, "%g ", vectorSum_h[i]);
	printf("\n");
	*/
```

```
	cudaFree(vector1_d);
	cudaFree(vector2_d);
	cudaFree(vectorSum_d);
	free(vectorSum_h);
	return elapsedTime;
}

int main() {
	float min_time, max_time, avg_time;
	min_time = max_time = avg_time = testingThreadsOfDevice(32); //запустить тест с 32 потоками на блок
	for (int i = 64; i <= 1024; i += 32) {
		float new_time = testingThreadsOfDevice(i);
		if (new_time > max_time)
			max_time = new_time;
		if (new_time < min_time)
			min_time = new_time;
		avg_time += new_time;
	}
	avg_time = avg_time / (1024 / 32);
	printf("time by Events: \n\t avg_time = %e min_time = %e max_time = %e\n",
		avg_time, min_time, max_time);



	return 0;
}
```

Скриншоты

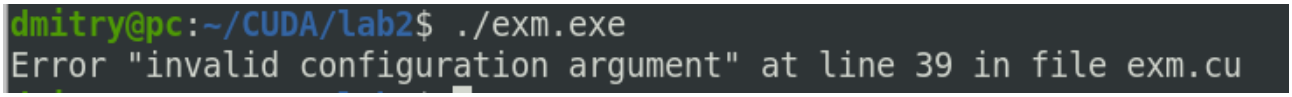

Рисунок 1 — выполнение программы с ошибкой

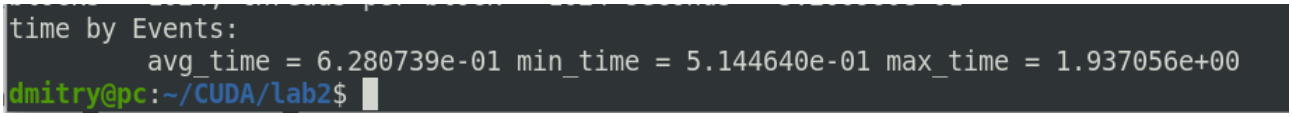

Рисунок 2 — выполнение программы из лабораторной 1, время измерено с помощью Events

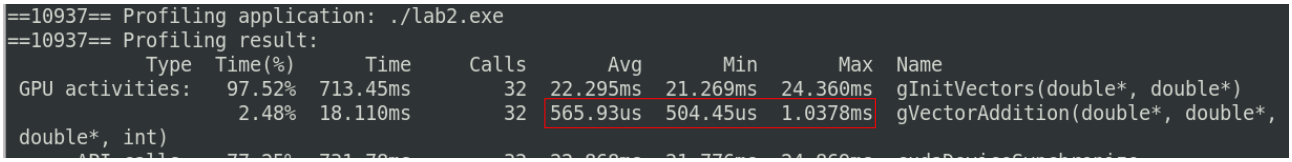

Рисунок 3 - выполнение программы из лабораторной 1, время измерено с помощью nvprof

Заключение

Я познакомился с тем, как обрабатывать ошибки при использовании CUDA и сравнил время выполнения сложения векторов через Events и nvprof. Время выполнения через Events оказалось больше из-за того, что в них используется дополнительные функции измерения времени, а в nvprof нет.