



МОСКОВСКИЙ АВИАЦИОННЫЙ
ИНСТИТУТ
НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ

Институт №8

«Компьютерные науки и прикладная математика»

Кафедра №806

«Вычислительная математика и программирование»

Отчет по лабораторной работе № 1,

по учебной дисциплине

«Параллельные и распределенные вычисления»

Выполнил:

Студент 1-го курса

Гр. М80-107М-22

Кузьмичев А. Н.

(подпись, дата)

Принял:

Кондратцев В. Л.

(подпись, дата)

Содержание

1. Постановка задачи.....	2
2. Описание решения	2
3. Аппаратное обеспечение и ПО	2
4. Основные моменты кода.....	2
5. Результат работы программы	3
6. Сравнение скорости выполнения на CPU и GPU.....	3
7. Выводы	4
8. GitHub.....	4
9. Исходный код.....	4

1. Постановка задачи

Вариант 7. Вычислить функцию тангенса.

2. Описание решения

Для нахождения тангенса используется встроенная в `math.h` функция `tanf()` - ей на вход подаются углы в радианах. В качестве массива исходных данных используется массив тестовых значений.

3. Аппаратное обеспечение и ПО

Видеокарта	Nvidia Geforce RTX 3060 6Gb
Процессор	Intel Core i5-11800H
IDE	VIM
ОС	Ubuntu 20.04 Focal

4. Основные моменты кода

В функции `main` идёт вызов и функции расчёта на GPU и на CPU. После завершения расчетов выводится время выполнения на GPU и CPU.

В функции `my_tan` реализован расчет на CPU. В функции `kernel` – на GPU.

5. Результат работы программы

```
1.150368
1.191754
1.234897
1.279942
1.327045
1.376382
1.428148
1.482561
1.539865
1.600335
1.664280
1.732051
1.804048
1.880726
1.962610
2.050304
2.144507
2.246037
2.355853
2.475086
2.605089
2.747477
2.904211
3.077684
3.270853
3.487414
3.732050
4.010781
4.331476
4.704631
5.144556
5.671280
6.313751
7.115369
8.144348
9.514368
11.430045
14.300658
19.081131
28.636255
57.290031
0.000000
0.017455
0.034921
0.052408
0.069927
0.087489
0.105104
0.122785
0.140541
0.158384
CPU time: 26.744000
GPU time: 0.333504
```

Рис.1. Вывод программы при N=1000000.

6. Сравнение скорости выполнения на CPU и GPU

При запуске программы видно, что вычисления на GPU производятся быстрее, чем на CPU.

Время выполнения программы при N = 1000000 :

N	GPU время выполнения, мс	CPU время выполнения, мс	t _{CPU} /t _{GPU}
1000000	0.333504	26.744000	80.1909422

7. Выводы

Для выполнения лабораторной работы №1 были реализованы программы для вычисления тангенса на CPU и GPU, а затем был проведен их сравнительный анализ.

8. GitHub

<https://github.com/pm-up/ParallelCalcs/tree/main/lab1>

9. Исходный код

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

static void CatchError(cudaError_t err, const char* file, int line) {
    if (err != cudaSuccess) {
        printf("%s in %s at line %d\n", cudaGetErrorString(err),
            file, line);
        exit(EXIT_FAILURE);
    }
}

#define CATCH_ERROR( err ) (CatchError( err, __FILE__, __LINE__ ))

__global__ void kernel(double *res, double *d_arr, long long int n)
{
    long long int tid = blockIdx.x * blockDim.x + threadIdx.x;
    long long int offset = blockDim.x * gridDim.x;

    while (tid < n) {
        // write result to array
        res[tid] = tanf( d_arr[tid%90] * 3.141592653589 / 180 );
        tid += offset;
    }
}

void my_tan(double *res, double *d_arr, long long int n) {
    long long int tid = 0;

    while (tid < n) {
        // write result to array
        res[tid] = tanf(d_arr[tid % 90] * 3.141592653589 / 180);
        tid += 1;
    }
}
```

```

    }
}

int main()
{
    long long int n = 1000000;

    cudaEvent_t time_of_start, time_of_end;
    float res_timer_gpu;
    CATCH_ERROR( cudaEventCreate( &time_of_start ) );
    CATCH_ERROR( cudaEventCreate( &time_of_end ) );

    double d_arr[90];    // array with degrees
    for (int i = 0; i < 90; i++) { // 0 to 90
        d_arr[i] = i;
    }

    double *res = (double*)malloc(n * sizeof(double));
    double *ar_d_dev, *res_dev;

    CATCH_ERROR( cudaMalloc( &res_dev, n * sizeof(double) ) );
    CATCH_ERROR( cudaMalloc( &ar_d_dev, 90 * sizeof(double) ) );

    CATCH_ERROR( cudaMemcpy( ar_d_dev, d_arr, 90 * sizeof(double),
        cudaMemcpyHostToDevice ) );

    CATCH_ERROR( cudaEventRecord( time_of_start ) );

    kernel <<<256,256>>>(res_dev, ar_d_dev, n);

    CATCH_ERROR( cudaEventRecord( time_of_end ) );
    CATCH_ERROR( cudaEventSynchronize( time_of_end ) );

    CATCH_ERROR( cudaEventElapsedTime( &res_timer_gpu, time_of_start,
        time_of_end ) );

    CATCH_ERROR( cudaEventDestroy( time_of_start ) );
    CATCH_ERROR( cudaEventDestroy( time_of_end ) );

    CATCH_ERROR( cudaMemcpy( res, res_dev, n * sizeof(double),
        cudaMemcpyDeviceToHost ) );

    CATCH_ERROR( cudaFree( ar_d_dev ) );
    CATCH_ERROR( cudaFree( res_dev ) );

    for (long long i = 0; i < n; i++) {
        printf("%f\n", res[i]);
    }

    free(res);

    res = (double*)malloc(n * sizeof(double));

```

```
double res_timer_cpu = 0.0;
clock_t begin = clock();
my_tan(res, d_arr, n);
clock_t end = clock();
res_timer_cpu += (double)(end - begin) / CLOCKS_PER_SEC;
printf("CPU time: %f\n", res_timer_cpu*1000);
printf("GPU time: %f\n", res_timer_gpu);
free(res);
return 0;
}
```