МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Кафедра вычислительной математики и программирования

**спецкурс «Параллельные и распределенные вычисления»**

**ОТЧЕТ**

**Лабораторная работа № 1**

**«Освоение программного обеспечения среды программирования NVIDIA»**

Выполнил: Каратаева Е.С.

Группа: М8О-114М-22, **вар. 9**

Преподаватель: Семенов С. А.

Москва, 2022

Содержание

[1. Постановка задачи 2](#_Toc52570380)

[2. Описание решения 2](#_Toc52570381)

[3. Аппаратное обеспечение и ПО 2](#_Toc52570382)

[4. Основные моменты кода 2](#_Toc52570383)

[5. Результат работы программы 3](#_Toc52570384)

[6. Сравнение скорости выполнения на CPU и GPU 7](#_Toc52570385)

[7. Выводы 8](#_Toc52570386)

[8. Приложения 8](#_Toc52570387)

# 1. Постановка задачи

Вариант 9. Вычислить функцию косинуса.

# 2. Описание решения

При нахождении косинуса использовалась встроенная в math.h функция cosf() - ей на вход подаются углы в радианах. В программе в качестве массива исходных данных использовался массив с тестовыми значениями в градусах, для вычисления производился перевод между единицами измерения.

# 3. Аппаратное обеспечение и ПО

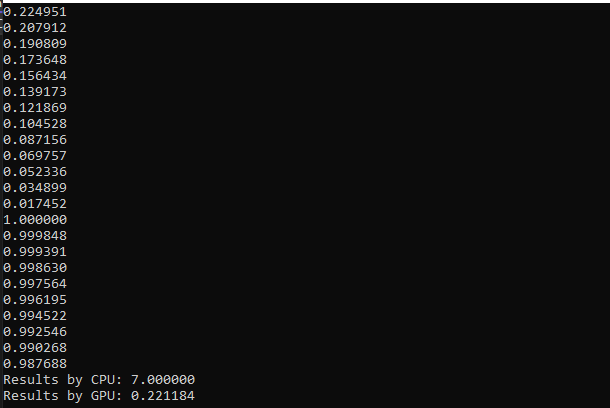
|  |  |
| --- | --- |
| Видеокарта | Nvidia Geforce RTX 3060 6Gb |
| Процессор | Intel Core i7-11800H |
| IDE | Visual Studio 2022 |
| OC | Windows 10 Pro 21H2 |

# 4. Основные моменты кода

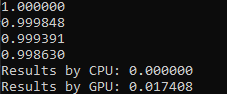
Исходный код представлен в личном репозитории Github (ссылка в Приложении А) и в приложении Б. В функции main идёт вызов и функции расчёта на GPU и на CPU. Время, занимаемое расчётами, выводится в конце после всех рассчитанных элементов.

А именно, функция my\_cos предназначена для расчетов на CPU. Функции kernel – на GPU. Код представлен при n = 1000000.

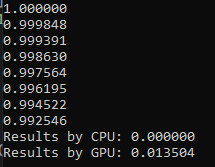
# 5. Результат работы программы



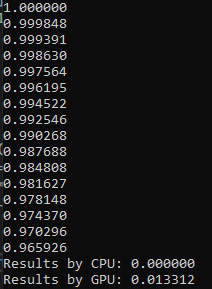
**Рис.1.** Вывод программы при N=1000000 в командной строке Windows (последние строки).



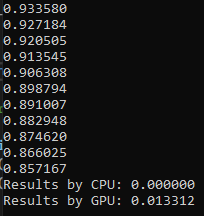
**Рис.2.** Вывод программы при N=4.



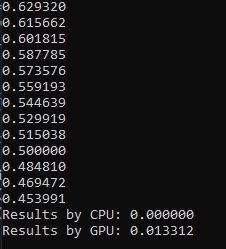
**Рис.3.** Вывод программы при N=8.



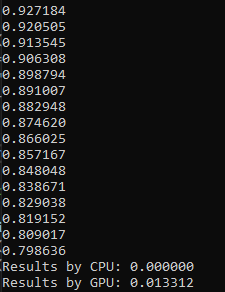
**Рис.4.** Вывод программы при N=16.



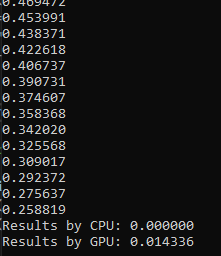
**Рис.5.** Вывод программы при N=32.



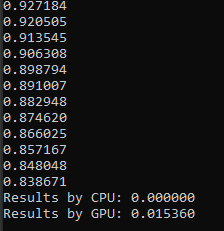
**Рис.6.** Вывод программы при N=64.



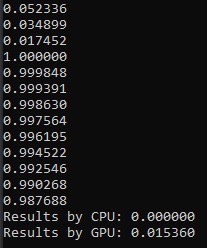
**Рис.7.** Вывод программы при N=128.



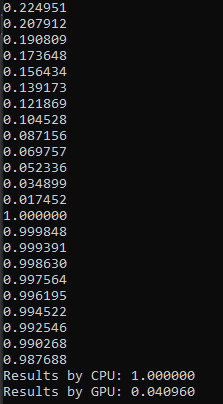
**Рис.8.** Вывод программы при N=256.



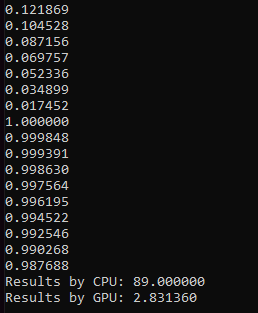
**Рис.9.** Вывод программы при N=1024.



**Рис.10.** Вывод программы при N=10000.



**Рис.10.** Вывод программы при N=100000.



**Рис.11.** Вывод программы при N=10000000.

# 6. Сравнение скорости выполнения на CPU и GPU

При запуске программы с различными значениями *N* видно, что вычисления на видеокарте произвелись быстрее, чем на процессоре компьютера, при *N*>100000. Для наглядности были выбраны значения 10000, 100000,1000000, 10000000.

Время выполнения программы при различных значениях *N*:

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **GPU**  **время выполнения, мс** | **CPU**  **время выполнения, мс** | **tCPU/tGPU** |
| 4 | 0,017408 | 0 | 0 |
| 8 | 0,013504 | 0 | 0 |
| 16 | 0,013312 | 0 | 0 |
| 32 | 0,013312 | 0 | 0 |
| 64 | 0,013312 | 0 | 0 |
| 128 | 0,013312 | 0 | 0 |
| 256 | 0,014336 | 0 | 0 |
| 1024 | 0,015360 | 0 | 0 |
| 10000 | 0,015360 | 0 | 0 |
| 100000 | 0,040960 | 1 | 24,4140625 |
| 1000000 | 0,221184 | 7 | 31,6478588 |
| 10000000 | 2,831360 | 89 | 31,43365732 |

**Рис.12.** График зависимости времени выполнения программы от порядка матрицы *N*.

# 7. Выводы

В Лабораторной работе №1 проведен анализ работы различных программ по решению задачи вычисления функции косинуса.

Во второй части Лабораторной работы было проведено сравнение времени выполнения программ на CPU и GPU.

# 8. Приложения

Приложение А. Ссылка на github.

<https://github.com/SimplyEle/cuda_labs/tree/main/lab1>

Приложение Б. Исходный код kernel.cu.

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.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 \*array\_degrees, 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) {

// writing to array result of calculations with conversion to degrees

res[tid] = cosf( array\_degrees[tid%90] \* 3.141592653589 / 180 );

tid += offset;

}

}

void my\_cos(double \*res, double \*array\_degrees, long long int n) {

long long int tid = 0;

while (tid < n) {

// writing to array result of calculations with conversion to degrees

res[tid] = cosf(array\_degrees[tid % 90] \* 3.141592653589 / 180);

tid += 1;

}

}

int main()

{

int count;

CATCH\_ERROR( cudaGetDeviceCount( &count ) );

if (count == 0) {

printf("there is no cuda device");

return -1;

}

long long int n = 1000000;

// for timer

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 array\_degrees[90]; // array with degrees

for (int i = 0; i < 90; i++) { // from 0 to 90

array\_degrees[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, array\_degrees, 90 \* sizeof(double), cudaMemcpyHostToDevice ) );

CATCH\_ERROR( cudaEventRecord( time\_of\_start ) ); // start of timer (GPU)

kernel <<<256,256>>>(res\_dev, ar\_d\_dev, n);

CATCH\_ERROR( cudaEventRecord( time\_of\_end )); // end of timer (GPU)

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\_cos(res, array\_degrees, n);

clock\_t end = clock();

res\_timer\_cpu += (double)(end - begin) / CLOCKS\_PER\_SEC;

printf("Results by CPU: %f\n", res\_timer\_cpu\*1000);

printf("Results by GPU: %f\n", res\_timer\_gpu);

free(res);

return 0;

}