МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Кафедра вычислительной математики и программирования

**спецкурс «Параллельные и распределенные вычисления»**

**ОТЧЕТ**

**Лабораторная работа № 1**

**«Программирование CUDA»**

Выполнил: Савельев А.С.

Группа: М80-107-22

Преподаватель: Семенов С. А.

Москва, 2022

Содержание

[1. Постановка задачи 2](#_Toc7492)

[2. Описание решения 2](#_Toc16227)

[4. Основные моменты кода 2](#_Toc5215)

[5. Результат работы программы 3](#_Toc10579)

[7. Выводы 4](#_Toc8193)

[8. Приложения 4](#_Toc23729)

# 1. Постановка задачи

Вариант 12: задать массив чисел и вычислить их квадраты

# Описание решения

Результатом выполнения программы является выходной массив *out* квадратов значений входного массива *in*

# Основные моменты кода

Реализация алгоритма на GPU и CPU выглядит следующим образом:

#define N 65536

\_\_global\_\_ void square(int \* in, int \* out)

{

out[blockIdx.x] = in[blockIdx.x] \* in[blockIdx.x];

}

void squareCPU(int\* in, int\* out)

{

for (int i = 0; i < N; i++)

{

out[i] = in[i] \* in[i];

}

}

В виду простоты алгоритма видимость каких-либо временных интервалов возможна лишь при многоразовым исполнении программы на CPU и GPU. для этих целей вызов устройства и вызов функции выполняются многократно с помощью цикла for:

double executionTime;

int iterations = 1000;

// Исполнение на устройстве

clock\_t startGPU = clock();

for (int i = 0; i < iterations; i++)

{

square <<<N, 1 >>> (dev\_in, dev\_out);

}

executionTime = ((double)clock() - startGPU) / CLOCKS\_PER\_SEC;

printf("\nGPU Execution time is %.10lf", executionTime);

// Исполнение на хосте

clock\_t startСPU = clock();

for (int i = 0; i < iterations; i++)

{

squareCPU(in, out);

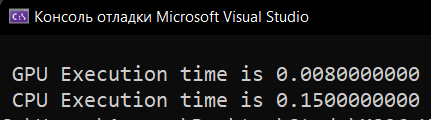
}

executionTime = ((double)clock() - startСPU) / CLOCKS\_PER\_SEC;

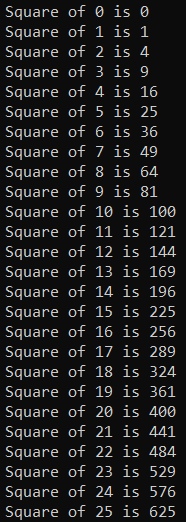
printf("\nCPU Execution time is %.10lf", executionTime);

# 5. Результат работы программы

# 



7. Выводы



В Лабораторной работе №1 проведен анализ работы различных программ по решению задачи возведения в квадрат значений массива. Из результата видно, что рост производительности GPU относительно CPU коррелирует с ростом размера обрабатываемого массива значений.

# Приложения

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

#include <stdio.h>

#include <time.h>

#define N 2048

\_\_global\_\_ void square(int \* in, int \* out)

{

out[blockIdx.x] = in[blockIdx.x] \* in[blockIdx.x];

}

void squareCPU(int\* in, int\* out)

{

for (int i = 0; i < N; i++)

{

out[i] = in[i] \* in[i];

}

}

int main()

{

// Инициализация массивов

int in[N], out[N];

int\* dev\_in, \* dev\_out;

// Выделение памяти

cudaMalloc((void\*\*)&dev\_in, N \* sizeof(int));

cudaMalloc((void\*\*)&dev\_out, N \* sizeof(int));

// Заполнение входного массива

for (int i = 0; i < N; i++)

{

in[i] = i;

}

// Передача данных устройству

cudaMemcpy(dev\_in, in, N \* sizeof(N), cudaMemcpyHostToDevice);

// Иницализация переменных мониторинга производительности

double executionTime;

int iterations = 100;

// Исполнение на устройстве

clock\_t startGPU = clock();

for (int i = 0; i < iterations; i++)

{

square <<<N, 1 >>> (dev\_in, dev\_out);

}

executionTime = ((double)clock() - startGPU) / CLOCKS\_PER\_SEC;

printf("\n GPU Execution time is %.60lf", executionTime);

// Исполнение на хосте

clock\_t startСPU = clock();

for (int i = 0; i < iterations; i++)

{

squareCPU(in, out);

}

executionTime = ((double)clock() - startСPU) / CLOCKS\_PER\_SEC;

printf("\n CPU Execution time is %.60lf", executionTime);

// Передача данных хосту

//cudaMemcpy(out, dev\_out, N \* sizeof(N), cudaMemcpyDeviceToHost);

// Вывод результирующих значений

//for (int i = 0; i < N; i++)

//{

// printf("\nsquare of %d is %d", i, out[i]);

//}

// Освобождение памяти

cudaFree(dev\_in);

cudaFree(dev\_out);

return 0;

}