МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Кафедра вычислительной математики и программирования

**спецкурс «Параллельные и распределенные вычисления»**

**ОТЧЕТ**

**Лабораторная работа**

**(дополнительное задание)**

**«Перемножение матриц»**

Выполнил: Сотников И. Д.

Группа: М8О-207М-20

Преподаватель: Семенов С. А.

Москва 2022

Содержание

[1. Постановка задачи 3](#_Toc52570380)

[2. Описание решения 3](#_Toc52570381)

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

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

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

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

[7. Выводы 8](#_Toc52570386)

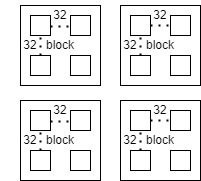
[8. Приложения 9](#_Toc52570387)

# 1. Постановка задачи

Вычислить произведение двух квадратных матриц.

# 2. Описание решения

Программа будет предлагать ввести размер квадратной матрицы, затем будут сгенерированы две матрицы заданного размера, которые будут перемножаться. Перемножение на девайсе будет исполняться двумерной матрицей нитей в блоке, если размер матрицы будет больше возможного числа нитей в блоке, будут выделены дополнительные блоки.



Каждая нить будет вычислять свою ячейку результирующей матрицы.

# 3. Аппаратное обеспечение и ПО

Процессор: AMD Ryzen 5 3600 (6 core, 12 thread, 3.6 GHz);

RAM: 16 GB;

Графический процессор: Nvidia GeForce GTX 460 SE (

Total amount of global memory: 1024 MB

(6) Multiprocessors, (48) CUDA Cores/MP: 288 CUDA Cores

GPU Max Clock rate: 1460 MHz

Memory Clock rate: 1700 MHz

Memory Bus Width: 256-bit);

CUDA Driver Version / Runtime Version: 8.0 / 8.0;

CUDA Capability Major/Minor version number: 2.1.

# 4. Основные моменты кода

В зависимости от размера матрицы выставляем число нитей и блоков. Так как максимальное количество нитей в блоке – 1024, если размер матрицы больше чем 32 х 32, выделяем дополнительные блоки, вычисляя их кол-во как seal(N / 32) где seal – функция приведения к ближайшему целому в большую сторону:

|  |
| --- |
| **Функция случайной величины** |
| void matrixMultiplication(float \*A, float \*B, float \*C, int N) {  int SIZE = N \* N;  dim3 threadsPerBlock(N, N);  dim3 blocksPerGrid(1, 1);  if (SIZE > 1024) {  threadsPerBlock.x = 32;  threadsPerBlock.y = 32;  blocksPerGrid.x = ceil(double(N) / double(threadsPerBlock.x));  blocksPerGrid.y = ceil(double(N) / double(threadsPerBlock.y));  }  matrixMultiplicationKernel <<<blocksPerGrid, threadsPerBlock>>>(A, B, C, N);  } |

В ядре идет вычисление ячейки (столбца и колонки) результирующей матрицы, для которой нить будет считать значение. Если координаты нити выходят за пределы размера матрицы (при размере не кратном 32), она не вычисляется из-за условия if.

|  |
| --- |
| **Ядро, выполняющееся на device:** |
| \_\_global\_\_ void matrixMultiplicationKernel(float\* A, float\* B, float\* C, int N) {  int ROW = blockIdx.y\*blockDim.y + threadIdx.y;  int COL = blockIdx.x\*blockDim.x + threadIdx.x;  float tmpSum = 0;  if (ROW < N && COL < N) {  for (int i = 0; i < N; i++) {  tmpSum += A[ROW \* N + i] \* B[i \* N + COL];  }  }  C[ROW \* N + COL] = tmpSum;  } |

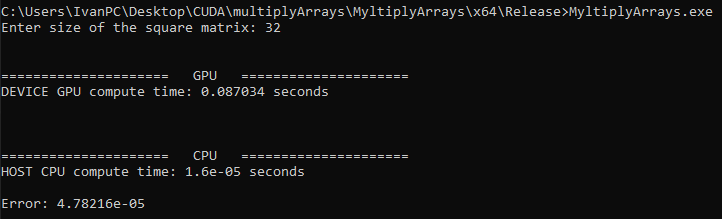
Матрицы будут представлены в виде векторов, где все строки склеены в один вектор последовательно сверху вниз. В качестве значений будут браться значения синусов и косинусов:

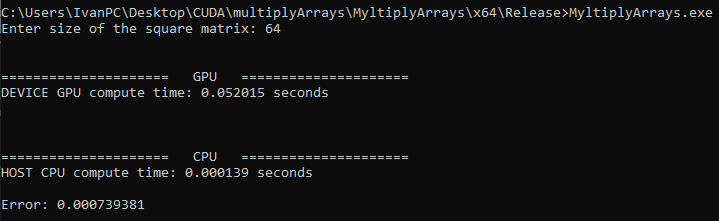
|  |
| --- |
| **Генерация двух матриц:** |
| int SIZE = N\*N;  vector<float> h\_A(SIZE);  vector<float> h\_B(SIZE);  for (int i = 0; i < N; i++) {  for (int j = 0; j < N; j++) {  h\_A[i\*N + j] = sin(i);  h\_B[i\*N + j] = cos(j);  }  } |

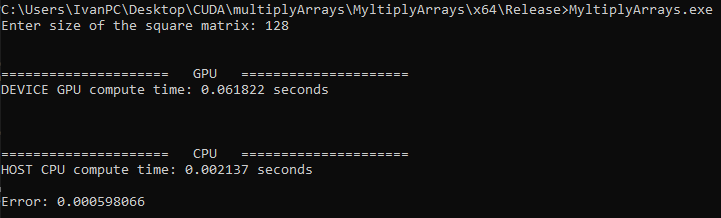
|  |
| --- |
| **Вычисление на CPU:** |
| int SIZE = N \* N;  float \*cpu\_C;  cpu\_C = new float[SIZE];  float sum;  for (int row = 0; row < N; row++) {  for (int col = 0; col < N; col++) {  sum = 0.f;  for (int n = 0; n < N; n++) {  sum += h\_A[row \* N + n] \* h\_B[n \* N + col];  }  cpu\_C[row \* N + col] = sum;  }  } |

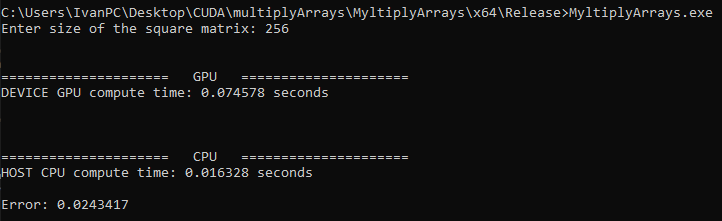
# 5. Результат работы программы

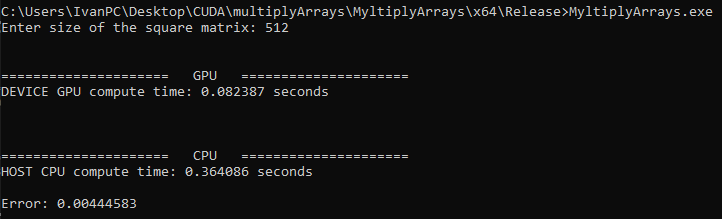
На вход программы подается количество строк (или столбцов) квадратных матриц, которых нужно перемножить:

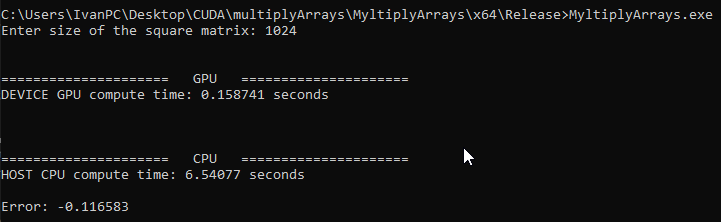


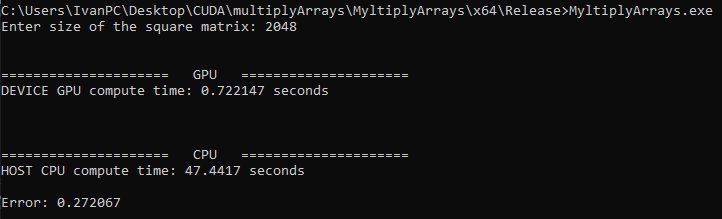


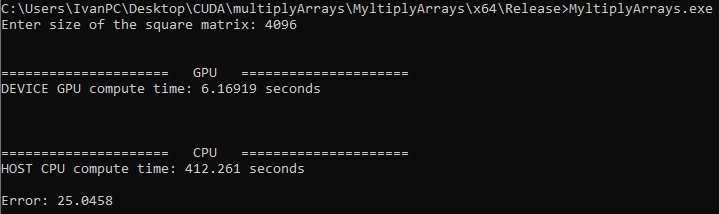










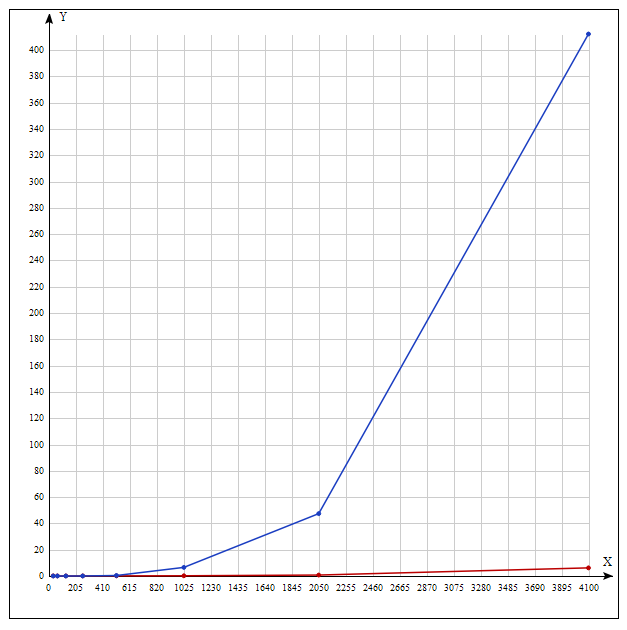


# 6. Сравнение скорости выполнения на CPU и GPU

При запуске программы с различными значениями *N* видно, что с повышением *N* время вычисления на процессоре возрастает экспоненциально, а на видеокарте увеличивается не сильно.

Время выполнения программы при различных значениях *N*:

|  |  |  |
| --- | --- | --- |
| **N** | **GPU**  **время выполнения, сек** | **CPU**  **время выполнения, сек** |
| 32 | 0.087 | 1.6 \* 10-5 |
| 64 | 0.052 | 1.4 \* 10-4 |
| 128 | 0.062 | 2.2 \* 10-3 |
| 256 | 0.075 | 0.012 |
| 512 | 0.082 | 0.364 |
| 1024 | 0.159 | 6.541 |
| 2048 | 0.722 | 47.442 |
| 4096 | 6.169 | 412.261 |



**Рис.** График зависимости времени выполнения программы от размера входных данных *N*, (красный цвет – GPU, синий цвет – CPU)

# 7. Выводы

В Лабораторной работе написана программа расчета произведения двух квадратных матриц, выполняющаяся на CPU и GPU соответственно. Был проведен анализ времени работы и сравнение двух реализаций подсчета на CPU и GPU.

# 8. Приложения

Проект Visual Studio: https://github.com/L0F1/CUDA\_labs