|  |  |
| --- | --- |
|  | **Правительство Российской Федерации**  **Федеральное государственное автономное образовательное учреждение высшего образования  «Национальный исследовательский университет «Высшая школа экономики»** |

**Отчёт по домашнему заданию №1 по курсу**

**«Высокопроизводительные Вычисления»**

**Тема работы: «Технология OpenMP»**

Выполнил:  
Забурунов Л. В., МСМТ221

(подпись)

Проверил(а):  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

4 октября 2022 г.

г. Москва

Оглавление

[Цель работы 3](#_Toc116330103)

[Постановка задачи 3](#_Toc116330104)

[1. Разработка базового решения 4](#_Toc116330105)

[2. Разработка оптимизированного решения 8](#_Toc116330106)

[3. Проверка работоспособности разработанных решений 9](#_Toc116330107)

[4. Анализ производительности разработанных решений 11](#_Toc116330108)

# Цель работы

Ознакомиться с особенностями многопоточного программирования на примере задачи умножения матриц с использованием технологии *OpenMP.*

# Постановка задачи

Требуется:

1. Создать программу на языке *C/C++*, осуществляющую вычисление результата умножения двух матриц по стандартному алгоритму (*General Matrix Multiplication)*. Элементы матриц представлены в виде чисел с плавающей точкой с двойной точностью, а сами матрицы хранятся в памяти по столбцам (*column-major order*);
2. Провести анализ масштабируемости созданного решения;
3. Изменить программу с целью оптимизации вычислений на узлах суперкомпьютера «Харизма» НИУ ВШЭ;
4. Провести анализ масштабируемости оптимизированного решения.

# Разработка базового решения

Помимо непосредственно программы для умножения матриц создадим вспомогательные методы для работы с данными.

Во-первых, позволим задавать исходные матрицы несколькими разными способами:

1. Вводом через консоль

Здесь мы не используем никаких дополнительных аргументов командной строки, программа предложит нам задать обе матрицы.

Структура: *hw1.out*

1. Получением из файла

В аргументах командной строки задаётся путь до файлов, из которых берутся исходные данные. Если задать только один файл, то из него будут инициализированы обе матрицы.

Структура: *hw1.out filepath1 [filepath2]*

1. Генерацией случайных матриц;

В аргументах командной строки задаётся ключевое слово *random,* а также два опциональных параметра – левая и правая границы интервала. Программа попросит ввести с консоли размерности матриц.

Структура: *hw1.out random [minInclusive] [maxExclusive]*

1. Генерацией единичных матриц.

В аргументах командной строки задаётся ключевое слово *identity,* а также опциональный параметр – размер обеих матриц. Если числовой параметр не задан, то будут созданы матрицы размерностей 10.

Структура: *hw1.out identity [dim]*

Для работы со всеми четырьмя вариантами созданы функции получения матрицы:

// RU: Получение матрицы от пользователя

// EN: Reading user input to get matrix

double\* GetMatrixFromConsoleInput(int& rows, int& columns)

// RU: Чтение файла с матрицей

// EN: Reading file to get matrix

double\* GetMatrixFromFile(char \*path, int& rows, int& columns)

// RU: Создание матрицы генератором случайных чисел

// EN: Randomly generating matrix

double\* GetRandomMatrix(int& rows, int& columns, double minInclusive, double maxExclusive)

// RU: Создание едичной матрицы

// EN: Generating identity matrix

double\* GetIdentityMatrix(int dims)

Одним из требований к создаваемой программе является работа с матрицами в «*column-major order*»*-*представлении. Поскольку при вводе через консоль или получении из файла матрицы имеют «*row-major order*»*-*представление, мы создаём вспомогательную функцию для преобразования:

// RU: Смена представления матрицы, заданной массивом, с row-major order на column-major order

// EN: Converting matrix from row-major order to column-major order

double\* RowMajorToColumnMajor(double \*matrix, int rows, int columns)

Во-вторых, для представления матриц в удобном виде создаём функцию вывода в консоль:

// RU: Вывод матрицы на консоль в удобном виде

// EN: Nice-formatted matrix output

void PrintMatrix(double \*matrix, int rows, int columns)

Сама программа подсчёта результата будет представлена в нескольких видах – с использованием векторных операций и без, а также в полной версии *GEMM* со сложением и в частичной только с умножением:

// RU: Внешний метод C = a \* (A \* B) + b \* C

// EN:

void dgemm(int, int, int, double\*, double\*, double\*&, float = 1.0, float = 0.0);

// RU: Внешний метод C = a \* (A \* B) + b \* C (последовательная версия для проверки)

// EN:

void dgemm\_serial(int, int, int, double\*, double\*, double\*&, float = 1.0, float = 0.0);

// RU: Проверочный метод с последовательным умножением матриц

// EN:

void blas\_dgemm\_serial(int, int, int, double\*, double\*, double\*&);

// RU: Метод умножения матриц по заданному в требованиях к ДЗ интерфейсу

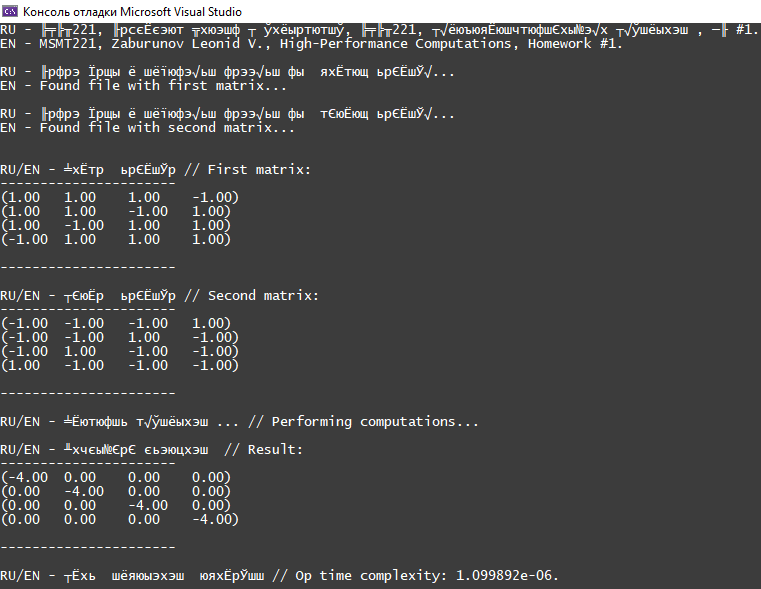
// EN:

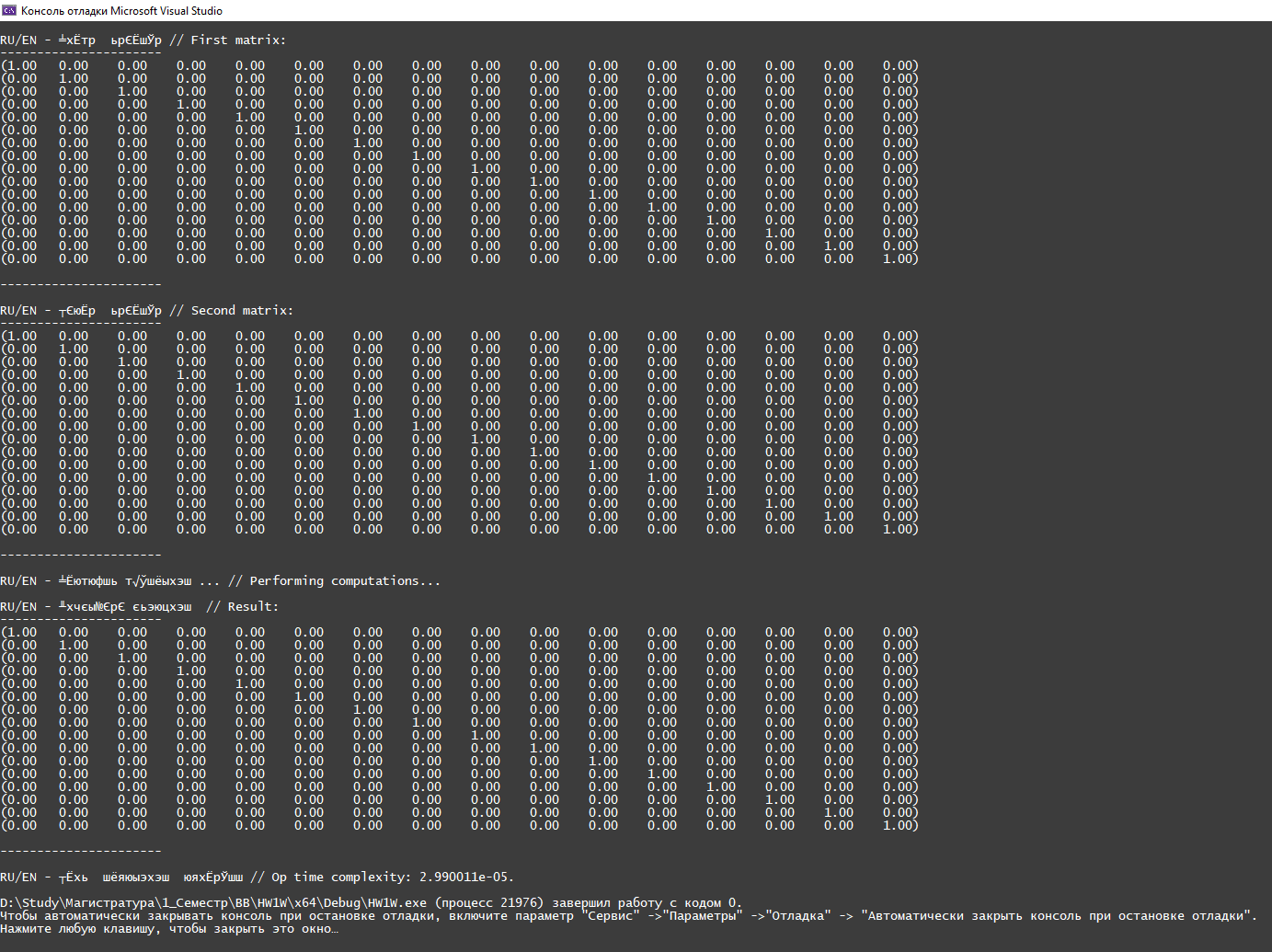
void blas\_dgemm(int, int, int, double\*, double\*, double\*&);

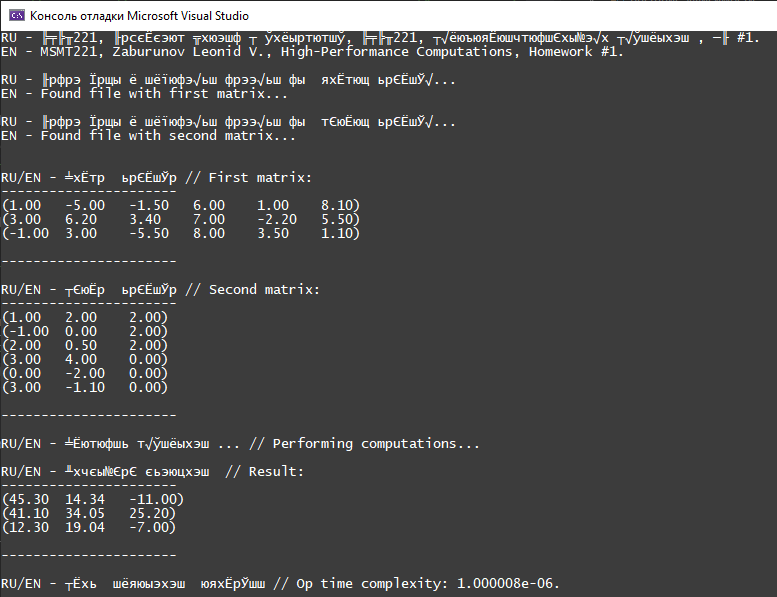
void dgemm\_scal(int, int, int, double \*, double \*, double \*&);

void dgemm\_vec(int, int, int, double \*, double \*, double \*&);

Проверим работоспособность программы на нескольких примерах с использованием домашнего компьютера:







# Разработка оптимизированного решения

В рамках доработки решения под узлы суперкомпьютера «Харизма» мы перепишем программу умножения матриц так, чтобы использовать векторные регистры в соответствии с *AVX512*:

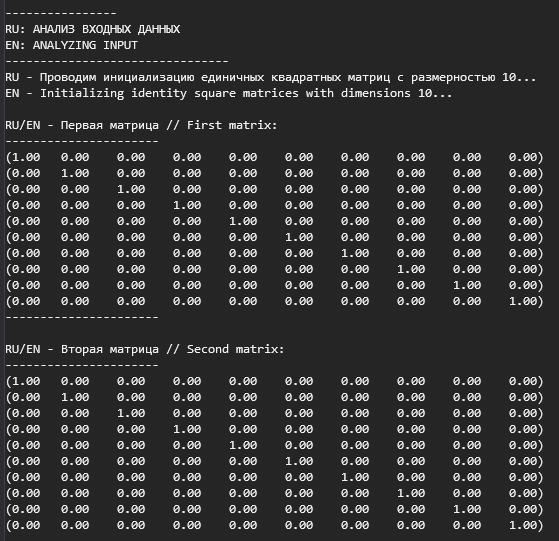
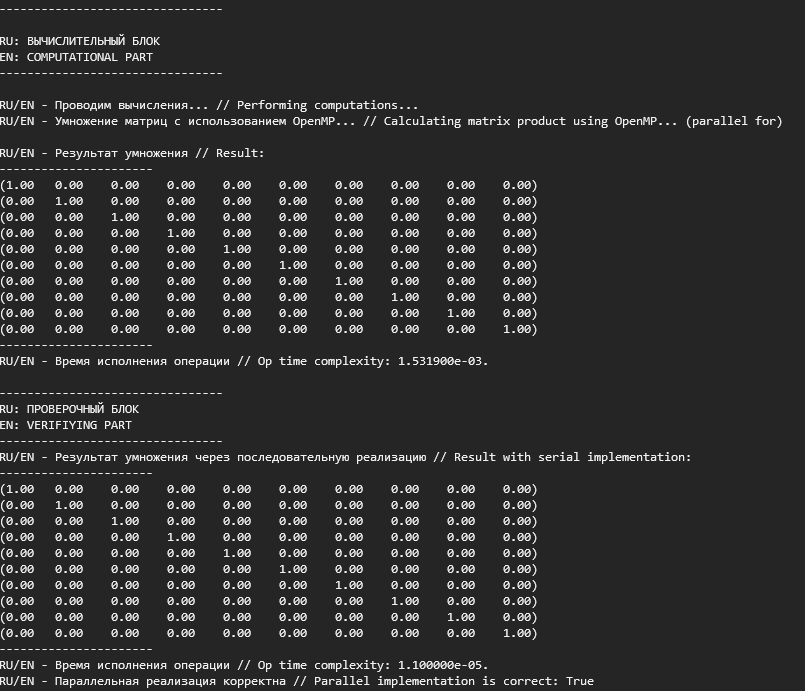
void dgemm\_avx(int, int, int, double \*, double \*, double \*&);

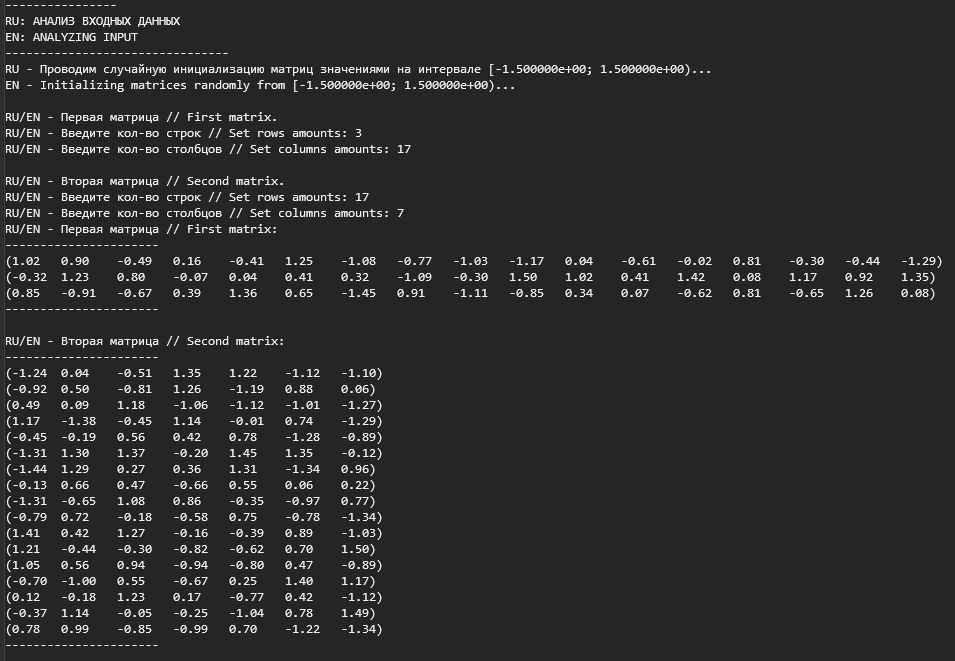
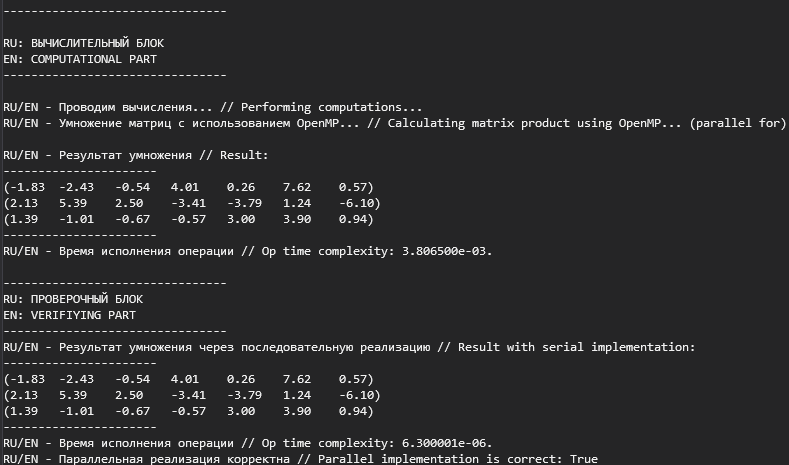
Векторные операции отлично подходят для задачи умножения матриц, поэтому в сочетании с многопоточной реализацией результат должен оказаться намного лучше. Каждый векторный регистр поместит 8 значений (формат *double* предполагает 64 бита на один элемент), мы используем 16 комплектов регистров в следующем виде:

* Двенадцать регистров используются для промежуточного хранения куска матрицы *C* размером 48x2 (то есть, по шесть регистров на двух столбцах);
* Два регистра используются для загрузки значений из матрицы *A*;
* Два регистра используются для загрузки значений из матрицы *B;*
* Матричное умножение осуществляется с помощью векторной операции *FMA* (*Fused Multiply-Add*), что избавляет нас от необходимости выделения регистров для временного хранения.

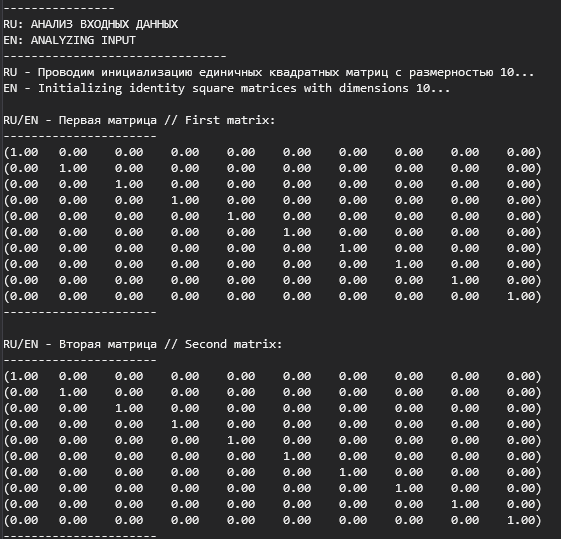
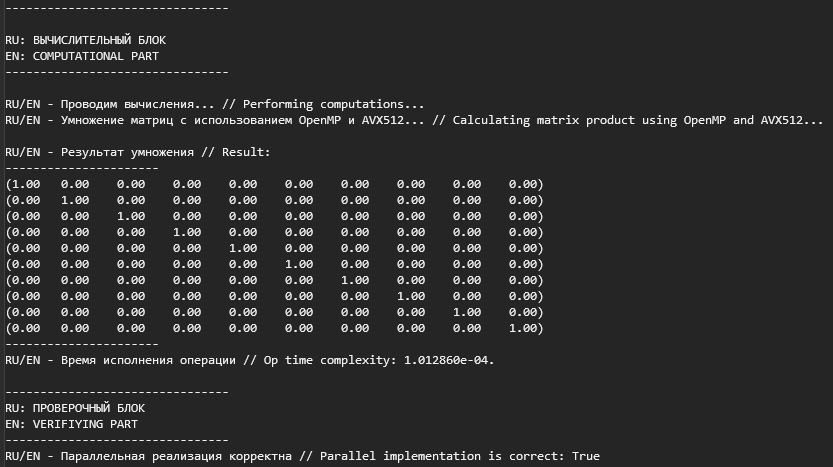
# Проверка работоспособности разработанных решений

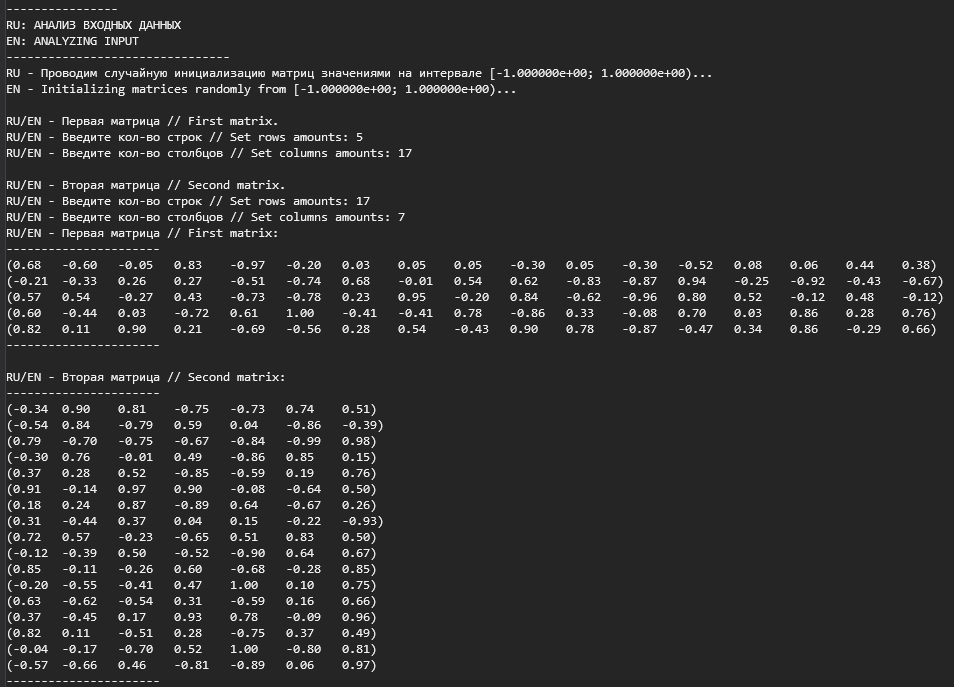
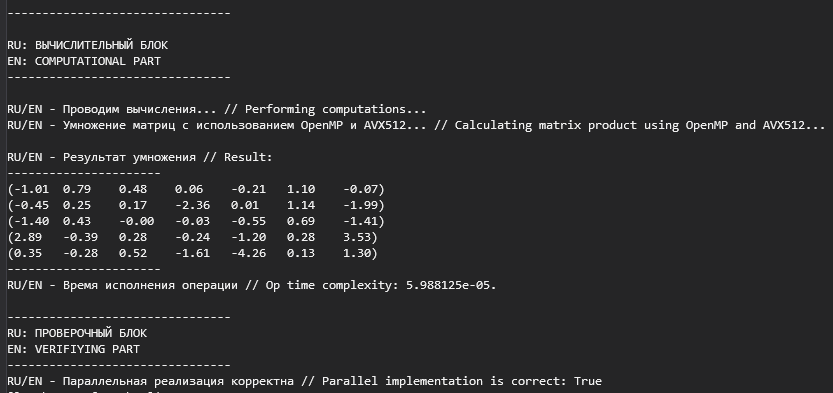
Проверим работоспособность с помощью сверки результатов с обыкновенной последовательной версией умножения матриц. Результаты проверки для версии, использующей только многопоточность:

И для версии, использующей *AVX-512*:

# Анализ производительности разработанных решений

Для анализа производительности будем запускать программу на квадратных единичных матрицах размерностей 400, 1000 и 2500. Для каждого размера будут представлены три графика:

* Диаграмма с усреднённым временем выполнения;
* Диаграмма с относительным временным ускорением от многопоточности;
* Диаграмма с относительным временным ускорением от использования *AVX*-инструкций.