МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации

**

**ОТЧЁТ**

**по лабораторной работе № 2**

**«**Оптимизация доступа к памяти»

**по дисциплине: «**Организация электронных вычислительных машин и вычислительных систем**»**

|  |  |
| --- | --- |
| Выполнил:  «2» \_\_мая\_\_\_ 2023 г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | Проверил:  «\_\_\_» \_\_\_\_\_\_ 2023 г.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Новосибирск 2023

Задание:

1. На языке С/С++/C# реализовать функцию DGEMM BLAS последовательное

умножение двух квадратных матриц с элементами типа double. Обеспечить

возможность задавать размерности матриц в качестве аргумента командной строки при запуске программы. Инициализировать начальные значения матриц случайными числами.

2. Провести серию испытаний и построить график зависимости времени

выполнения программы от объёма входных данных. Например, для квадратных

матриц с числом строк/столбцов 1000, 2000, 3000, … 10000.

3. Оценить предельные размеры матриц, которые можно перемножить на вашем

вычислительном устройстве.

4. Реализовать дополнительную функцию DGEMM\_opt\_1, в которой

выполняется оптимизация доступа к памяти, за счет построчного перебора

элементов обеих матриц.

5. Реализовать дополнительную функцию DGEMM\_opt\_2, в которой

выполняется оптимизация доступа к памяти, за счет блочного перебора элементов

матриц. Обеспечить возможность задавать блока, в качестве аргумента функции.

Определить размер блока, при котором достигается максимальное ускорение.

6. Реализовать дополнительную функцию DGEMM\_opt\_3, в которой

Выполняется оптимизация доступа к памяти, за счет векторизации кода.

7. Оценить ускорение умножения для матриц фиксированного размера,

например,1000х1000, 2000х2000, 5000х5000, 10000х10000.

8. С помощью профилировщика для исходной программы и каждого способа

оптимизации доступа к памяти оценить количество промахов при работе к КЭШ

памятью (cache-misses).

9. Подготовить отчет отражающий суть, этапы и результаты проделанной

работы.Ход работы:

Работу было решено выполнить на ЯП C++. Использовался компилятор GNU GCC, с указанием дополнительных аргументов:

g++ ./optimized.cpp -o ./optimized -O0 -Wall -g

Код для выполнения умножения матриц выглядит следующим образом:

int i, j, k;

    // Выполняем умножение матриц

    for (i = 0; i < size; i++) {

        for (j = 0; j < size; j++) {

            for (k = 0; k < size; k++) {

                result[i \* size + j] += a[i \* size + k] \* b[k \* size + j];

            }

        }

    }

Для того, чтобы была возможность задавать параметры и размерность матриц, в качестве аргументов главной функции программы добавляем аргументы командной строки:

int main(int argc, char \*argv[]) {

return 0;

}

Для инициализации случайными числами матриц, можно использовать генератор случайных чисел и распределение для этих чисел.

std::mt19937 gen(42);

std::uniform\_real\_distribution<double> dist(0.0, 1.0);

Таким образом, матрицы будут заполняться числами типа double в диапазоне [0.0, 1.0].

Проведем серию испытаний и засечем время выполнения функции умножения матриц. Размерности матриц выберем 100, 200, 300, … 1000.

Чтобы автоматизировать данные тесты и не делать ничего вручную, напишем bash скрипт, используя полученные знания из 1 ЛР, для запуска и проведения тестов, а также код на python для построения графиков с использованием библиотеки matplotlib.

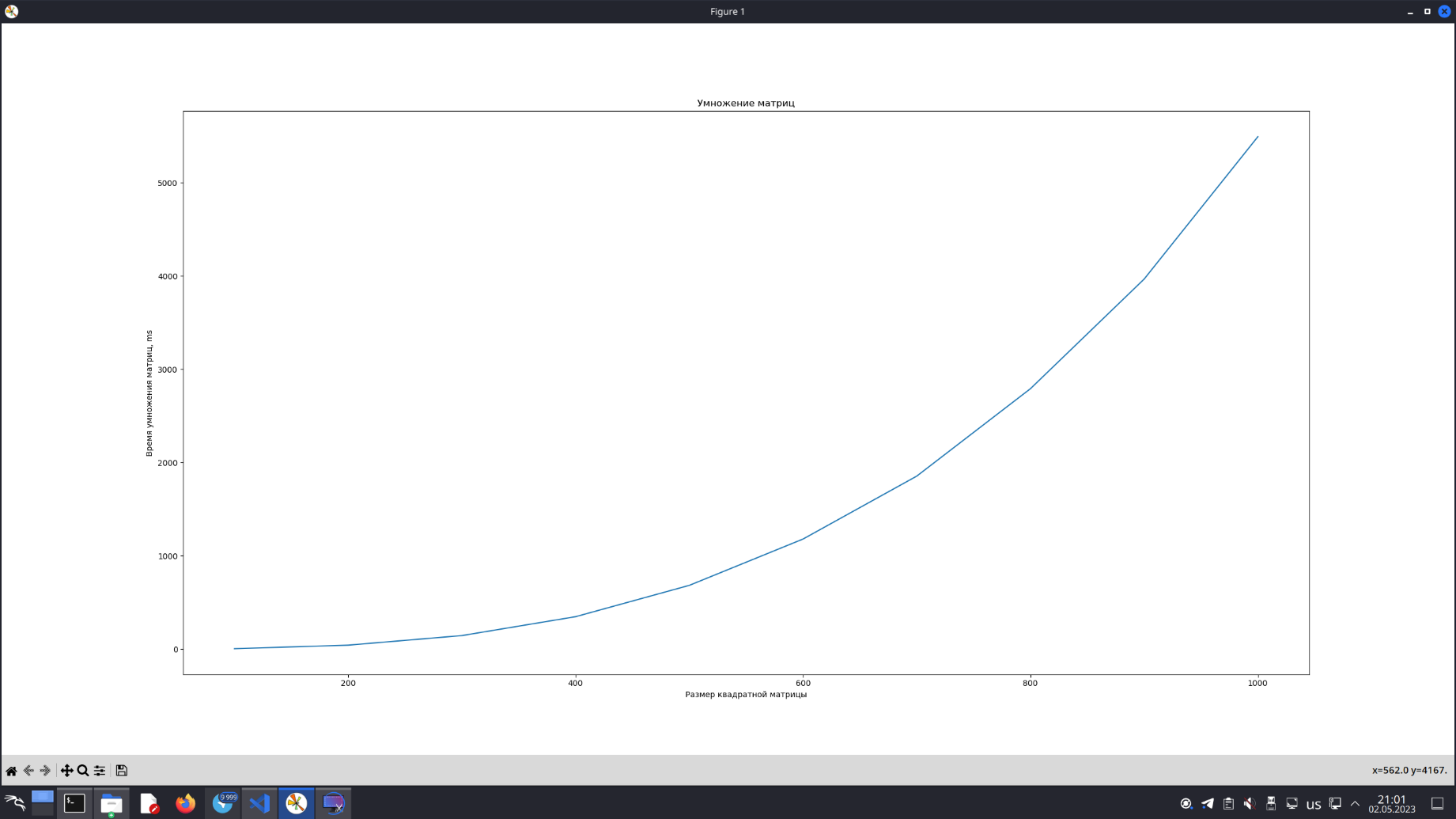


Рисунок 1 – График зависимости времени выполнения программы от размерности матриц

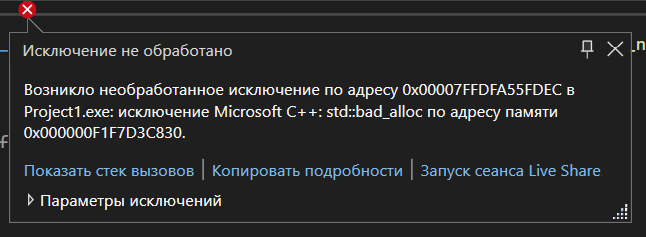
Оценим предельные размеры матриц:

предельный размер матрицы = sqrt(доступная оперативная память / (размер одного элемента \* количество матриц)) 65 к ломанулся.

Если учесть, что операционная система и другие процессы могут занимать до 4 ГБ памяти, то доступная память для перемножения матриц будет около 13 ГБ. Эту память нужно перевести в байты, так как один элемент double занимает 8 байт памяти в C++, то есть нужно умножить на 1024 в 3 степени. Количество матриц – 3.

Подставив значения, получим результат примерно в 24116 x 24116 элементов для одной матрицы. Проверим это.

При запуске кода с размерностью, больше указанной, происходит утечка памяти и происходит экстренное завершение работы программы (рисунок 2).

  
Рисунок 2 – Экстренное завершение работы программы

В качестве первой оптимизации умножения, изменим нашу первоначальную функцию умножения так, чтобы перебор элементов обеих матриц происходил построчно:

 int i, j, k;

    // Выполняем умножение матриц

    for (i = 0; i < size; i++) {

        for (k = 0; k < size; k++) {

            double aik = a[i \* size + k];

            for (j = 0; j < size; j++) {

                result[i \* size + j] += aik \* b[k \* size + j];

            }

        }

    }

Для следующей оптимизации напишем функцию с блочным перебором элементов матриц:

for (int i = 0; i < size; i += block\_size) {

        for (int j = 0; j < size; j += block\_size) {

            for (int k = 0; k < size; k += block\_size) {

                for (int i1 = i; i1 < i + block\_size && i1 < size; ++i1) {

                    for (int j1 = j; j1 < j + block\_size && j1 < size; ++j1) {

                        double sum = 0.0;

                        for (int k1 = k; k1 < k + block\_size && k1 < size; ++k1) {

                            sum += a[i1 \* size + k1] \* b[k1 \* size + j1];

                        }

                        result[i1 \* size + j1] += sum;

                    }

                }

            }

        }

    }

Переменную block\_size мы передаем в функцию в качестве аргумента, а задается она через аргументы командной строки:

else if (std::string(argv[i]).find("--opt2=") == 0) {

                type\_of\_func = 2;

                std::string num = std::string(argv[i]).substr(7);

                if(std::stoi(num) && std::stoi(num) > 0)

                    block\_size = std::stoi(num);

}

Теперь определим размер блока, при котором достигается максимальное ускорение. Для этого проведем серию испытаний. Модифицировав скрипт на bash и код на python, снова запускаем тесты.

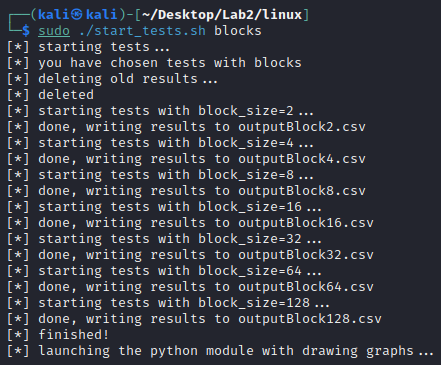


Рисунок 3 – Вывод скрипта при работе

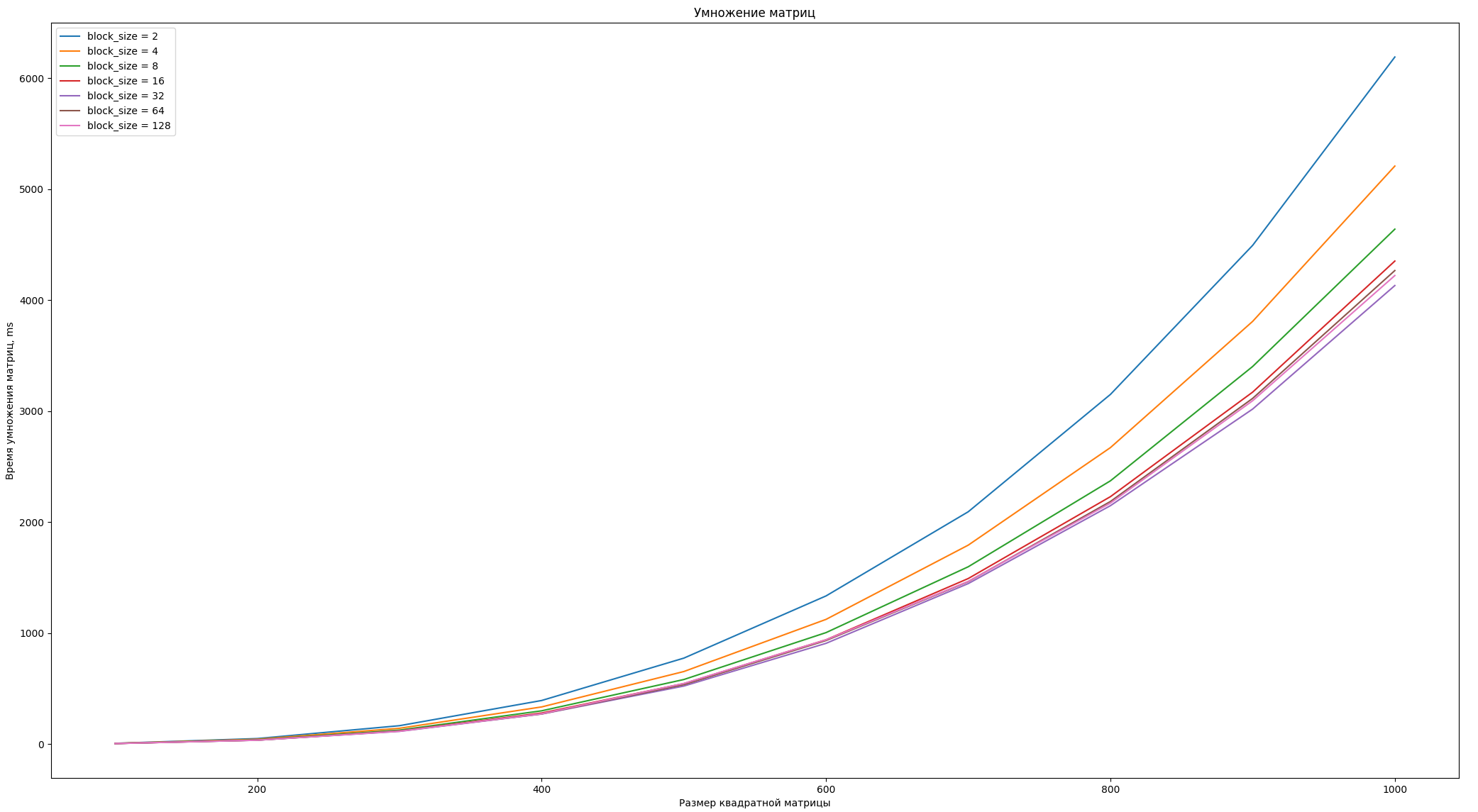


Рисунок 4 – Графики зависимости времени умножения матриц от их размерностей для разных размеров блоков

Из рисунка следует, что размер блока в 32 дает наибольший прирост в скорости вычислений.

Вместо реализации векторизации кода вручную, можно использовать автовекторизацию компилятором. Для этого нужно указать опцию -O3 компилятора GNU GCC:

g++ ./optimized.cpp -o ./optimized3 -O3 -Wall -g

Чтобы оценить ускорение умножения также воспользуемся написанными ранее bash скриптом и кодом на python.

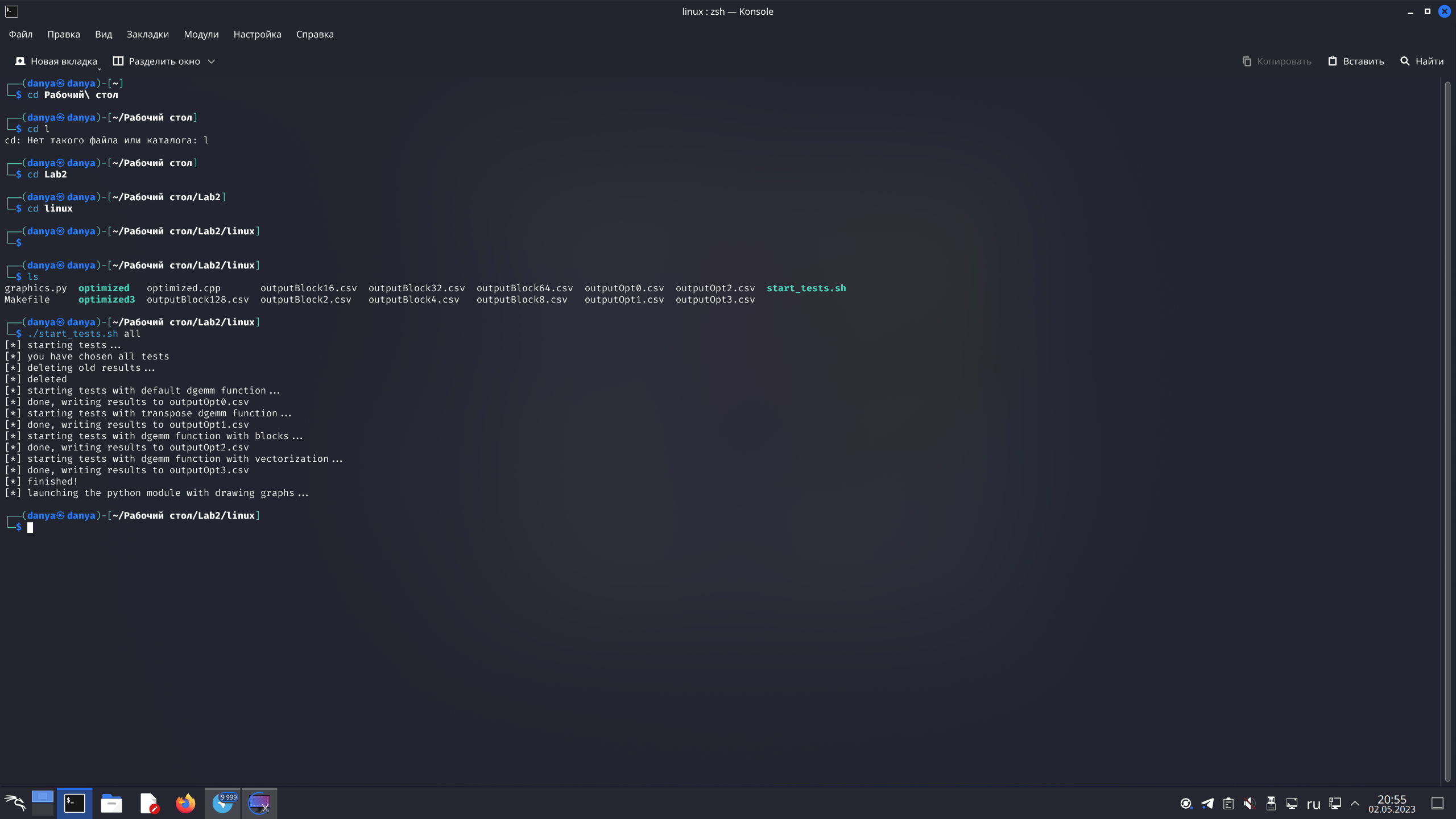


Рисунок 5 – Вывод скрипта при работе

Для вычислений с блочной функцией берем размер блока 32.

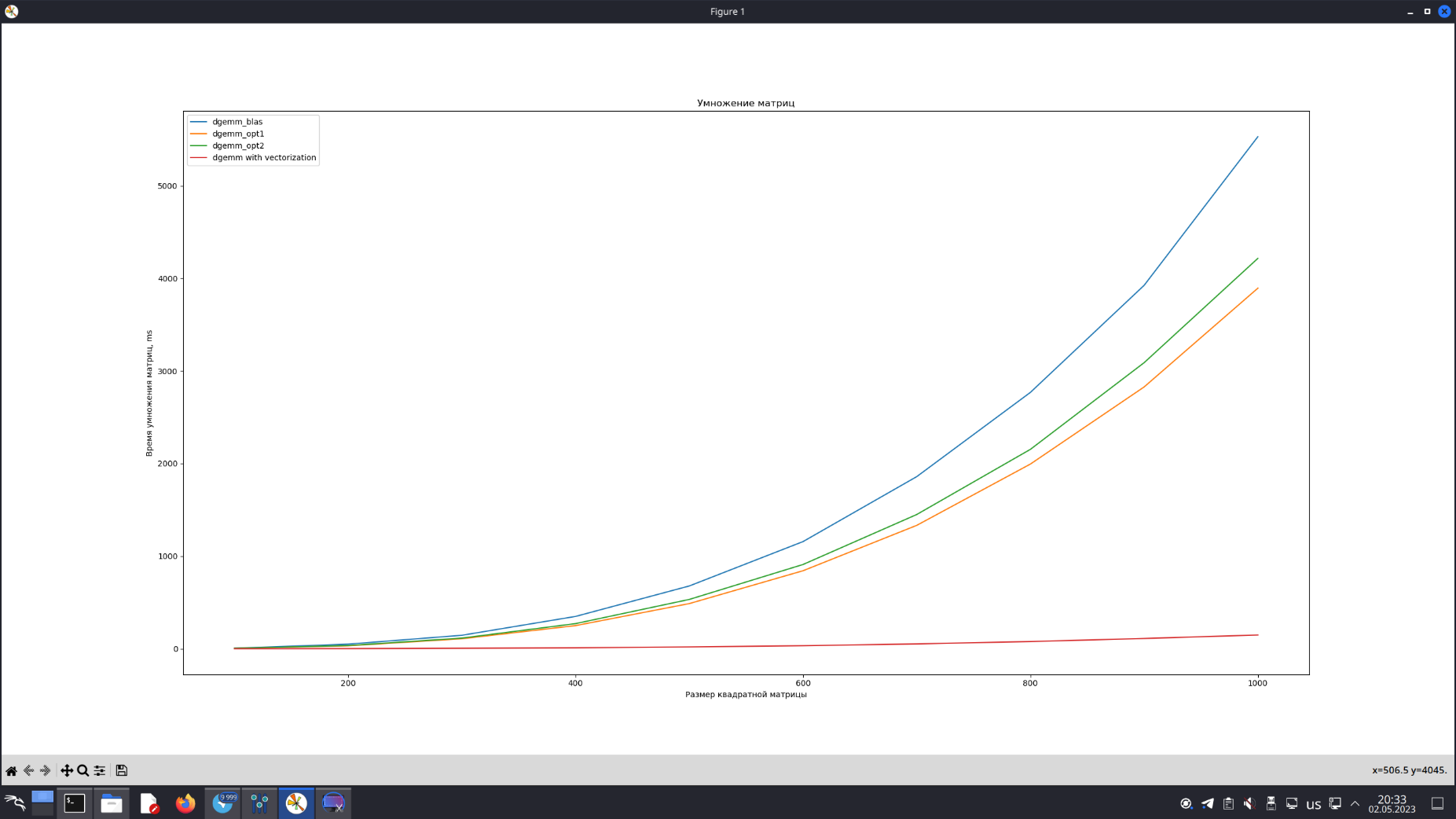


Рисунок 6 – Графики зависимости времени умножения матриц от их размерностей для всех функций разом

Как видно из рисунка, векторизация дала огромный прирост в скорости вычислений.

Теперь определим количество промахов при работе с КЭШ памятью (cache-misses).

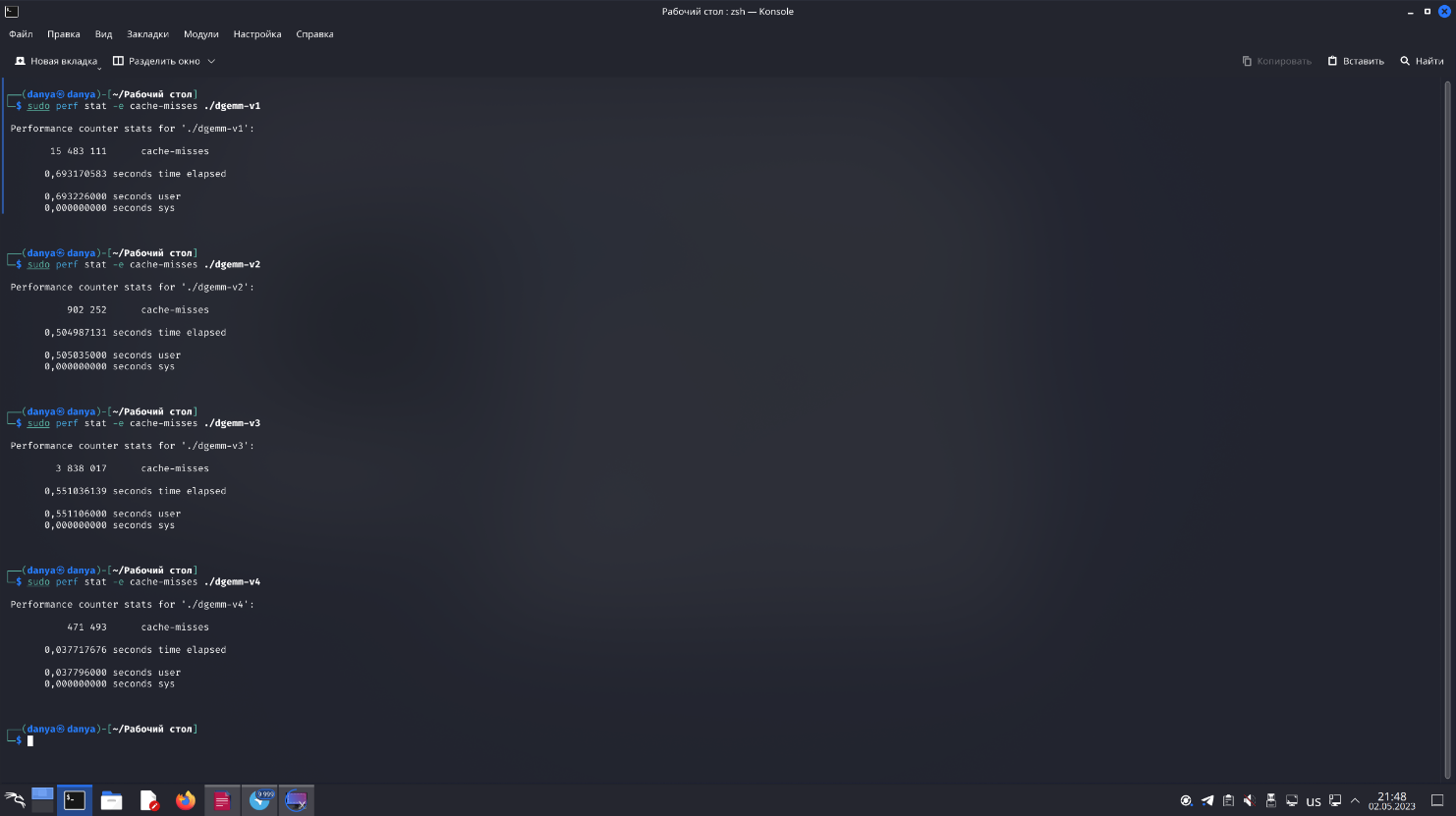


Рисунок 7 – Количество промахов для стандартной функции

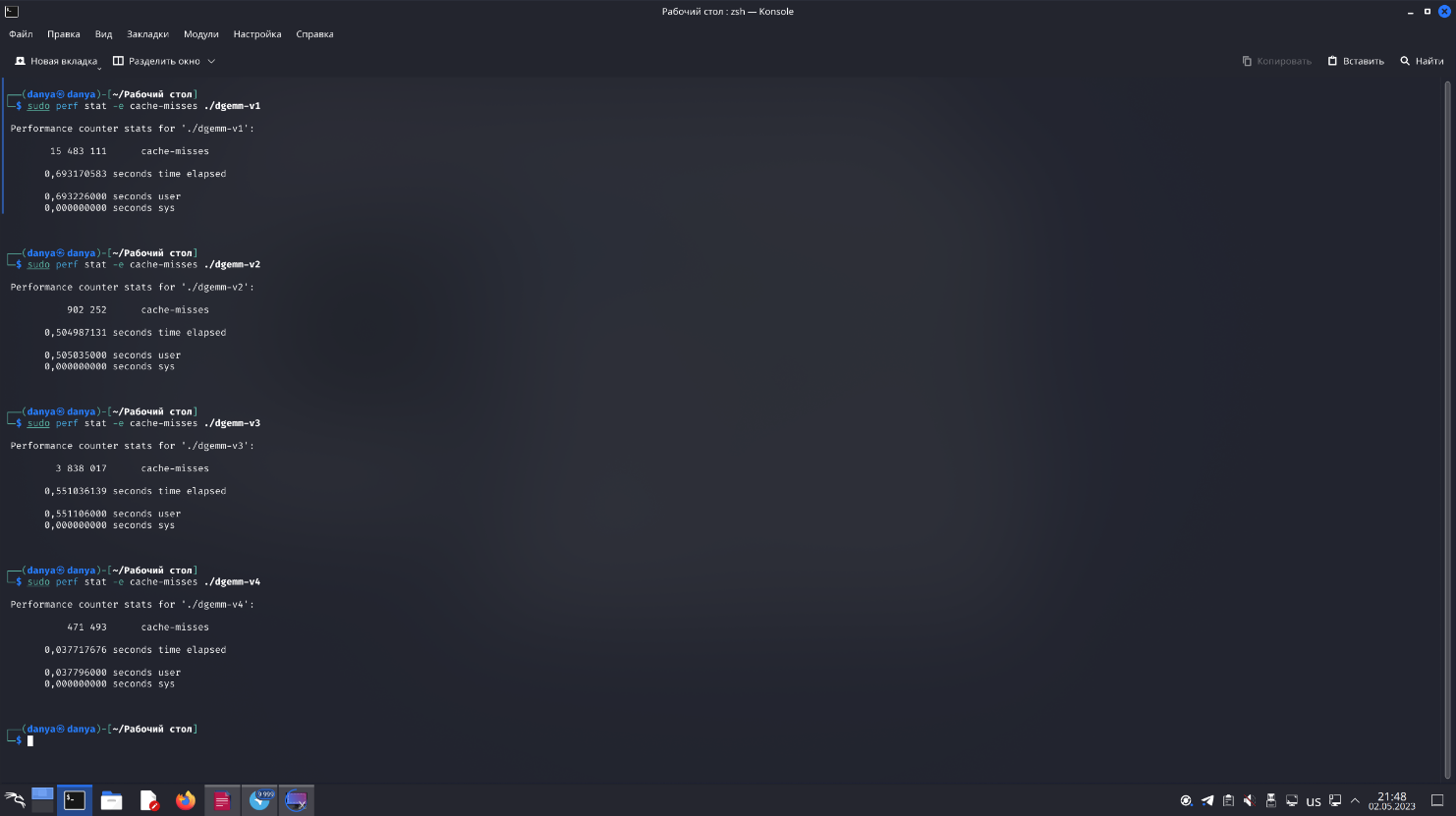


Рисунок 8 – Количество промахов для функции с построчным перебором элементов

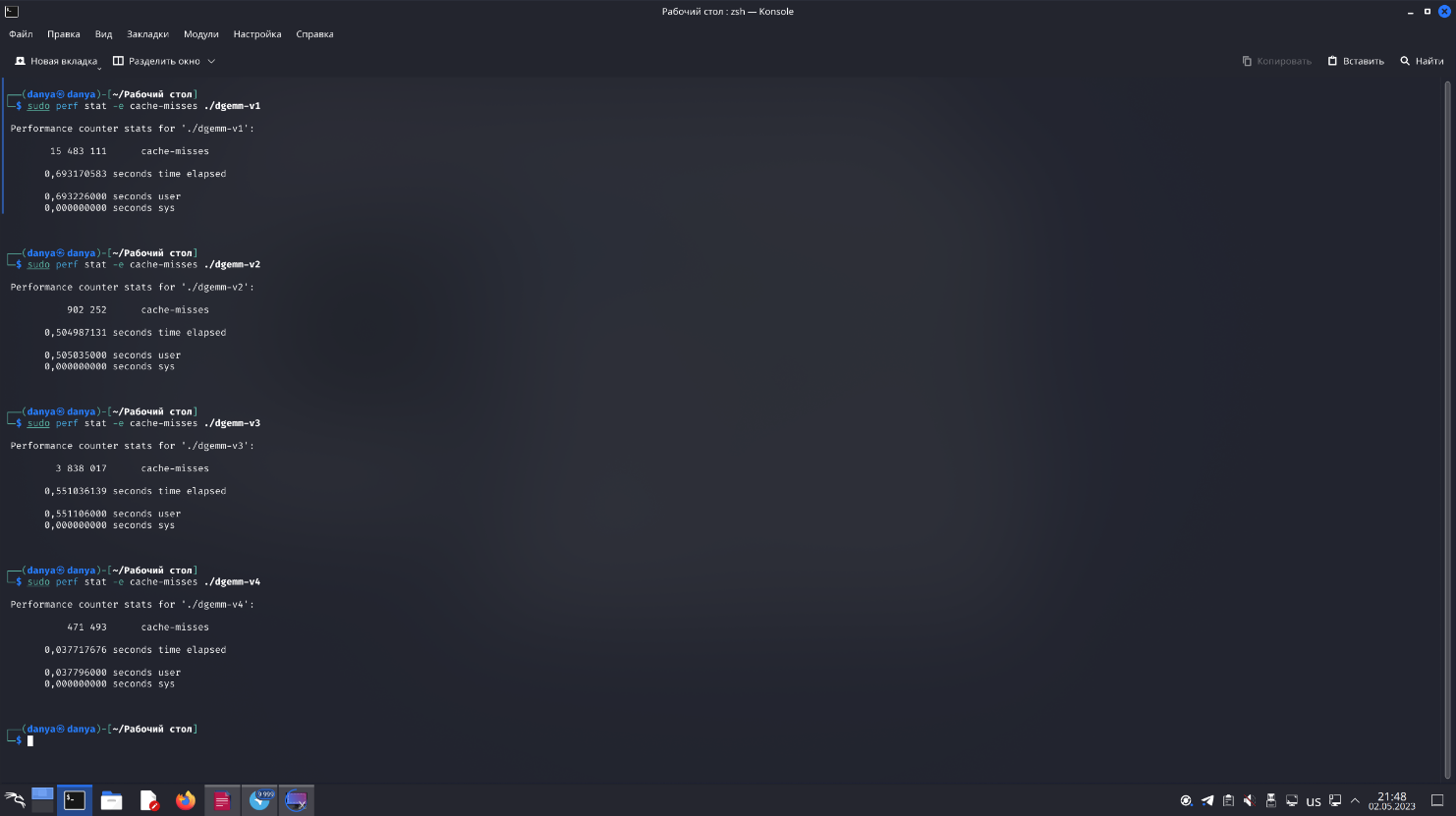


Рисунок 9 – Количество промахов для функции с блочным перебором элементов

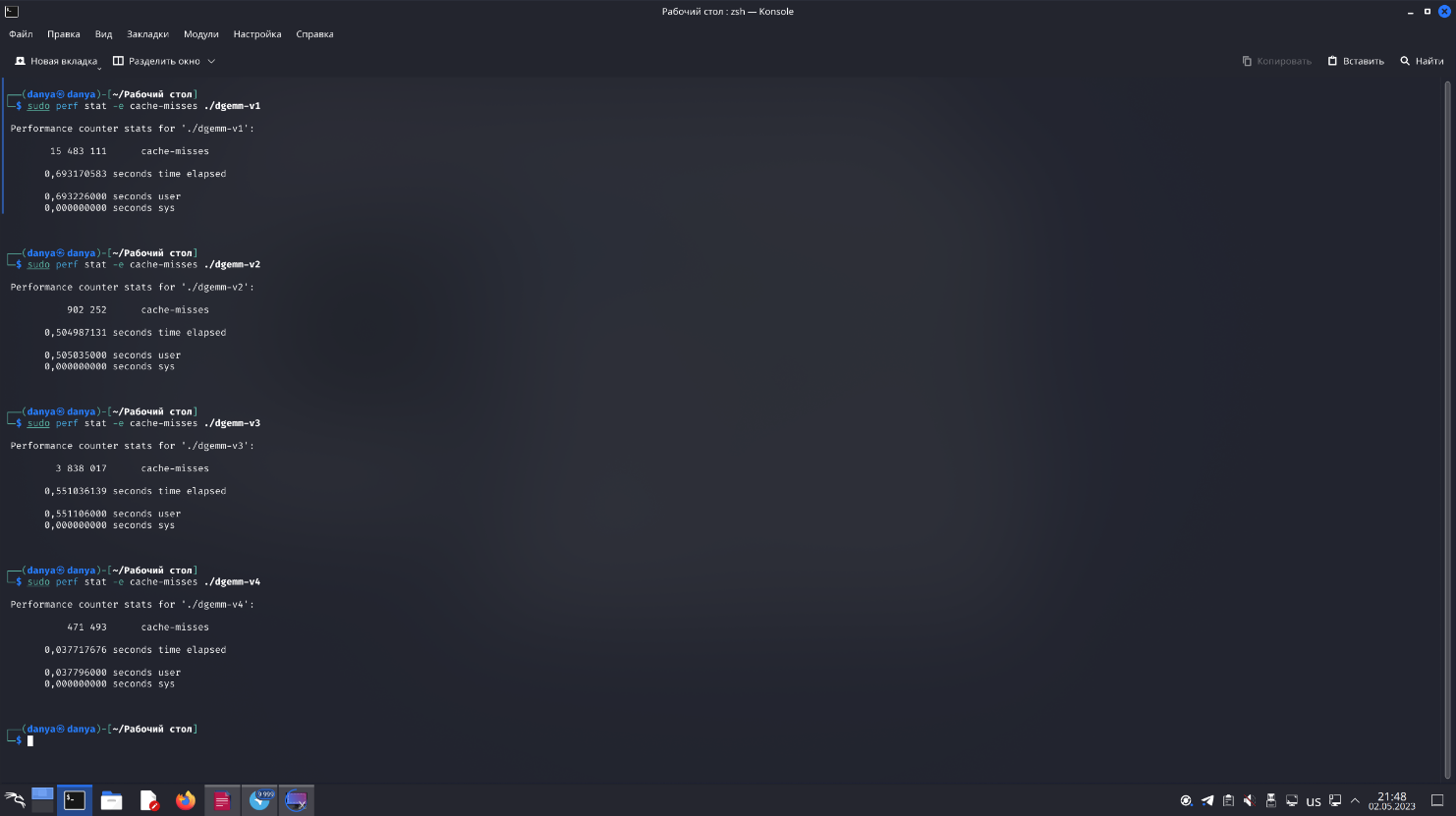


Рисунок 10 – Количество промахов для функции с автовекторизацией

Вывод: в ходе выполнения данной лабораторной работы я реализовал функции умножения матриц на таком языке программирования, как C++. Я научился оптимизировать вычисления, используя различные методы умножения матриц. Также я получил практический опыт в написании bash скриптов для автоматизации задач и научился работать с библиотекой matplotlib для python, с помощью которой были построены все графики.

ПРИЛОЖЕНИЕ

graphics.py

import matplotlib.pyplot as plt

import csv

import sys

plt.title("Умножение матриц")

plt.xlabel("Размер квадратной матрицы")

plt.ylabel("Время умножения матриц, ms")

if \_\_name\_\_ == "\_\_main\_\_":

    descriptions = {}

    if sys.argv[1] != "all" and sys.argv[1] != "blocks":

        print("[ERROR!] Unknown argument: %s\nAvailable args: 'all', 'blocks'" % sys.argv[1])

        sys.exit()

    else:

        if sys.argv[1] == "all":

            descriptions = {"outputOpt0.csv": "dgemm\_blas", "outputOpt1.csv": "dgemm\_opt1",

                            "outputOpt2.csv": "dgemm\_opt2", "outputOpt3.csv": "dgemm with vectorization"}

        else:

            nums = [2,4,8,16,32,64,128]

            for num in nums:

                descriptions.update({f"outputBlock{num}.csv": f"block\_size = {num}"})

    for filename, description in descriptions.items():

        x\_values = []

        y\_values = []

        with open(filename, 'r') as f:

            reader = csv.reader(f, delimiter=' ')

            for row in reader:

                if (row[0] and row[1]):

                    x\_values.append(float(row[0]))

                    y\_values.append(float(row[1]))

                else:

                    print('Error in file reading: empty params')

            f.close()

        plt.plot(x\_values, y\_values, label=description)

    plt.legend()

    plt.show()

Makefile

nonvect:

    g++ .\optimized.cpp -o .\optimized.exe -O0 -Wall -g

vect:

    g++ .\optimized.cpp -o .\optimized3.exe -O3 -Wall -g

start\_tests.sh

#!/bin/bash

sizes=(100 200 300 400 500 600 700 800 900 1000)

modes=("--opt0" "--opt1" "--opt2=32" "vect")

block\_sizes=(2 4 8 16 32 64 128)

filename=""

echo "[\*] starting tests..."

if [ "$1" == "all" ]; then

    echo "[\*] you have chosen all tests"

    echo "[\*] deleting old results..."

    filenames=("Opt0" "Opt1" "Opt2")

    for file in ${filenames[@]};

    do

        if [ -f output${file}.csv ]; then

            echo "da"

            rm output${file}.csv

        fi

    done

    echo "[\*] deleted"

    for mode in ${modes[@]};

    do

        if [ "${mode}" == "--opt0" ]; then

            echo "[\*] starting tests with default dgemm function..."

            filename="Opt0"

        elif [ "${mode}" == "--opt1" ]; then

            echo "[\*] starting tests with transpose dgemm function..."

            filename="Opt1"

        elif [ "${mode}" == --opt2=\* ]; then

            echo "[\*] starting tests with dgemm function with blocks..."

            filename="Opt2"

        elif [ "${mode}" == "vect" ]; then

            echo "[\*] starting tests with dgemm function with vectorization..."

            filename="Opt3"

        fi

        for size in ${sizes[@]};

        do

            program\_output=$(./optimized ${size} ${mode} -t)

            if [ "${mode}" == "vect" ]; then

                program\_output=$(./optimized3 ${size} --opt1 -t)

            fi

            time\_val=$(echo "${program\_output}" | cut -d' ' -f1)

            echo "${size} ${time\_val}" >> output${filename}.csv

        done

        echo "[\*] done, writing results to output${filename}.csv"

    done

elif [ "$1" == "blocks" ]; then

    echo "[\*] you have chosen tests with blocks"

    echo "[\*] deleting old results..."

    for block\_size in ${block\_sizes[@]};

    do

        if [ -f outputBlock${block\_size}.csv ]; then

            rm outputBlock${block\_size}.csv

        fi

    done

    echo "[\*] deleted"

    for block\_size in ${block\_sizes[@]};

    do

        echo "[\*] starting tests with block\_size=${block\_size}..."

        for size in ${sizes[@]};

        do

            program\_output=$(./optimized ${size} --opt2=${block\_size} -t)

            time\_val=$(echo "${program\_output}" | cut -d' ' -f1)

            echo "${size} ${time\_val}" >> outputBlock${block\_size}.csv

        done

        echo "[\*] done, writing results to outputBlock${block\_size}.csv"

    done

else

    echo "Incorrect argument! Use 'all' or 'blocks' instead"

    exit 1

fi

echo "[\*] finished!"

echo "[\*] launching the python module with drawing graphs..."

python3 graphics.py "$1"

optimized.cpp

#include <iostream>

#include <vector>

#include <iomanip>

#include <random>

#include <string>

#include <chrono>

void print\_matrix(const std::vector<double>& matrix, int size) {

    for (int i = 0; i < size; i++) {

        for (int j = 0; j < size; j++) {

            std::cout << std::setw(10) << matrix[i \* size + j] << " ";

        }

        std::cout << std::endl;

    }

}

void dgemm(const std::vector<double>& a, const std::vector<double>& b, std::vector<double> &result, const int size) {

    int i, j, k;

    // Выполняем умножение матриц

    for (i = 0; i < size; i++) {

        for (j = 0; j < size; j++) {

            for (k = 0; k < size; k++) {

                result[i \* size + j] += a[i \* size + k] \* b[k \* size + j];

            }

        }

    }

}

void dgemm\_opt1(std::vector<double>& a, std::vector<double>& b, std::vector<double> &result, const int size) {

    int i, j, k;

    // Выполняем умножение матриц

    for (i = 0; i < size; i++) {

        for (k = 0; k < size; k++) {

            double aik = a[i \* size + k];

            for (j = 0; j < size; j++) {

                result[i \* size + j] += aik \* b[k \* size + j];

            }

        }

    }

}

void dgemm\_opt2(const std::vector<double>& a, const std::vector<double>& b, std::vector<double>& result, const int size, const int block\_size) {

    for (int i = 0; i < size; i += block\_size) {

        for (int j = 0; j < size; j += block\_size) {

            for (int k = 0; k < size; k += block\_size) {

                for (int i1 = i; i1 < i + block\_size && i1 < size; ++i1) {

                    for (int j1 = j; j1 < j + block\_size && j1 < size; ++j1) {

                        double sum = 0.0;

                        for (int k1 = k; k1 < k + block\_size && k1 < size; ++k1) {

                            sum += a[i1 \* size + k1] \* b[k1 \* size + j1];

                        }

                        result[i1 \* size + j1] += sum;

                    }

                }

            }

        }

    }

}

int main(int argc, char \*argv[]) {

    if (argc < 2 || argc > 5) {

        std::cout << "Usage: " << argv[0] << " <matrix size> [-o] [-t] [--opt0] [--opt1] [--opt2=<block\_size>]" << std::endl;

        std::cout << "Help: \n"

        << "-o        print all matrix on screen\n"

        << "-t        timer\n"

        << "--opt0    default func dgemm blass\n"

        << "--opt1    optimization by line-by-line iteration of elements\n"

        << "--opt2    optimization due to block iteration of matrix elements, you can specify block size\n";

        return 0;

    }

    bool output = false;

    bool timer = false;

    int type\_of\_func = -1;

    int block\_size = 2;// размер блока задаем на случай, если не задаст пользователь

    for (auto i = 2; i < argc; i++) {

        if (std::string(argv[i]) == "-o")

            output = true;

        else if (std::string(argv[i]) == "-t")

            timer = true;

        else if (type\_of\_func == -1) {

            if (std::string(argv[i]) == "--opt0")

                type\_of\_func = 0;

            else if (std::string(argv[i]) == "--opt1")

                type\_of\_func = 1;

            else if (std::string(argv[i]).find("--opt2=") == 0) {

                type\_of\_func = 2;

                std::string num = std::string(argv[i]).substr(7);

                if(std::stoi(num) && std::stoi(num) > 0)

                    block\_size = std::stoi(num);

            }

        }

    }

    srand(time(NULL));

    int n = atoi(argv[1]);

    std::vector<double> a(n\*n);

    std::vector<double> b(n\*n);

    // генератор случайных чисел и распределение для этих чисел

    std::mt19937 gen(42);

    std::uniform\_real\_distribution<double> dist(0.0, 1.0);

    // заполнение случайными числами

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

            a[i \* n + j] = dist(gen);

            b[i \* n + j] = dist(gen);

        }

    }

    // Создаем пустую матрицу для результата

    std::vector<double> result(n\*n, 0.0);

    std::chrono::time\_point<std::chrono::system\_clock> start;

    std::chrono::time\_point<std::chrono::system\_clock> end;

    switch (type\_of\_func)

    {

    case 0: {

        start = std::chrono::system\_clock::now();

        dgemm(a, b, result, n);

        end = std::chrono::system\_clock::now();

        break;

    }

    case 1: {

        start = std::chrono::system\_clock::now();

        dgemm\_opt1(a, b, result, n);

        end = std::chrono::system\_clock::now();

        break;

    }

    case 2: {

        start = std::chrono::system\_clock::now();

        dgemm\_opt2(a, b, result, n, block\_size);

        end = std::chrono::system\_clock::now();

        break;

    }

    default:

        std::cout << "You didn\'t choose type of functions";

        exit(0);

    }

    //если задан флаг с подсчетом времени работы кода, то вычисляем его и выводим на экран

    if (timer)

        std::cout << std::chrono::duration <double, std::milli> (end-start).count() << " ms" << std::endl;

    if (output) {

        std::cout << "Matrix A:" << std::endl;

        print\_matrix(a,n);

        std::cout << "Matrix B:" << std::endl;

        print\_matrix(b,n);

        std::cout << "Matrix C = A \* B:" << std::endl;

        print\_matrix(result,n);

    }

    return 0;

}