Федеральное агентство связи

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра вычислительных систем

Отчет по лабораторной работе

по дисциплине «Архитектура вычислительных систем»

Лабораторная работа №4

«Оптимизация доступа к памяти»

Выполнил: студент 3 курса группы ИП-811 Мироненко К. А

Проверил: доцент кафедры ВС Ефимов А. В.

Оглавление

[1. Постановка задачи 3](#_Toc62080435)

[2. Примеры работы программы 5](#_Toc62080436)

[*Приложение* Листинг 10](#_Toc62080437)

# Постановка задачи

***Тема:*** оптимизация доступа к памяти.***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.

1. 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.
2. Оценить предельные размеры матриц, которые можно перемножить на вашем вычислительном устройстве.
3. Реализовать дополнительную функцию DGEMM\_opt\_1, в которой выполняется оптимизация доступа к памяти, за счет построчного перебора элементов обеих матриц.
4. \* 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.
5. \*\* Реализовать дополнительную функцию DGEMM\_opt\_3, в которой выполняется оптимизация доступа к памяти, за счет векторизации кода.
6. Оценить ускорение умножения для матриц фиксированного размера, например, 1000х1000, 2000х2000, 5000х5000, 10000х10000.

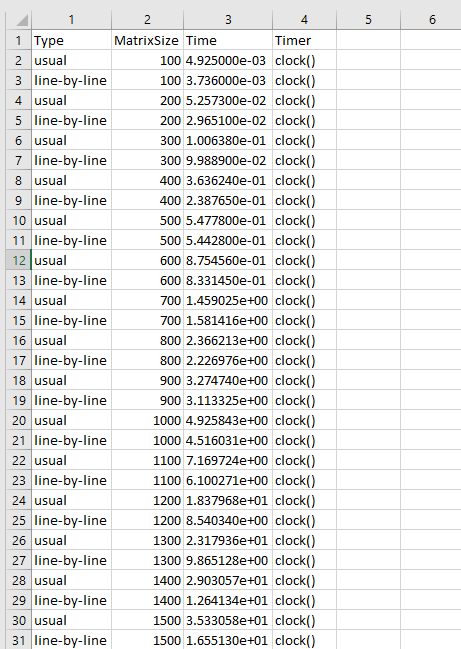
\* Для блочного умножения матриц определить размер блока, при котором достигается максимальное ускорение.

1. С помощью профилировщика для исходной программы и каждого способа оптимизации

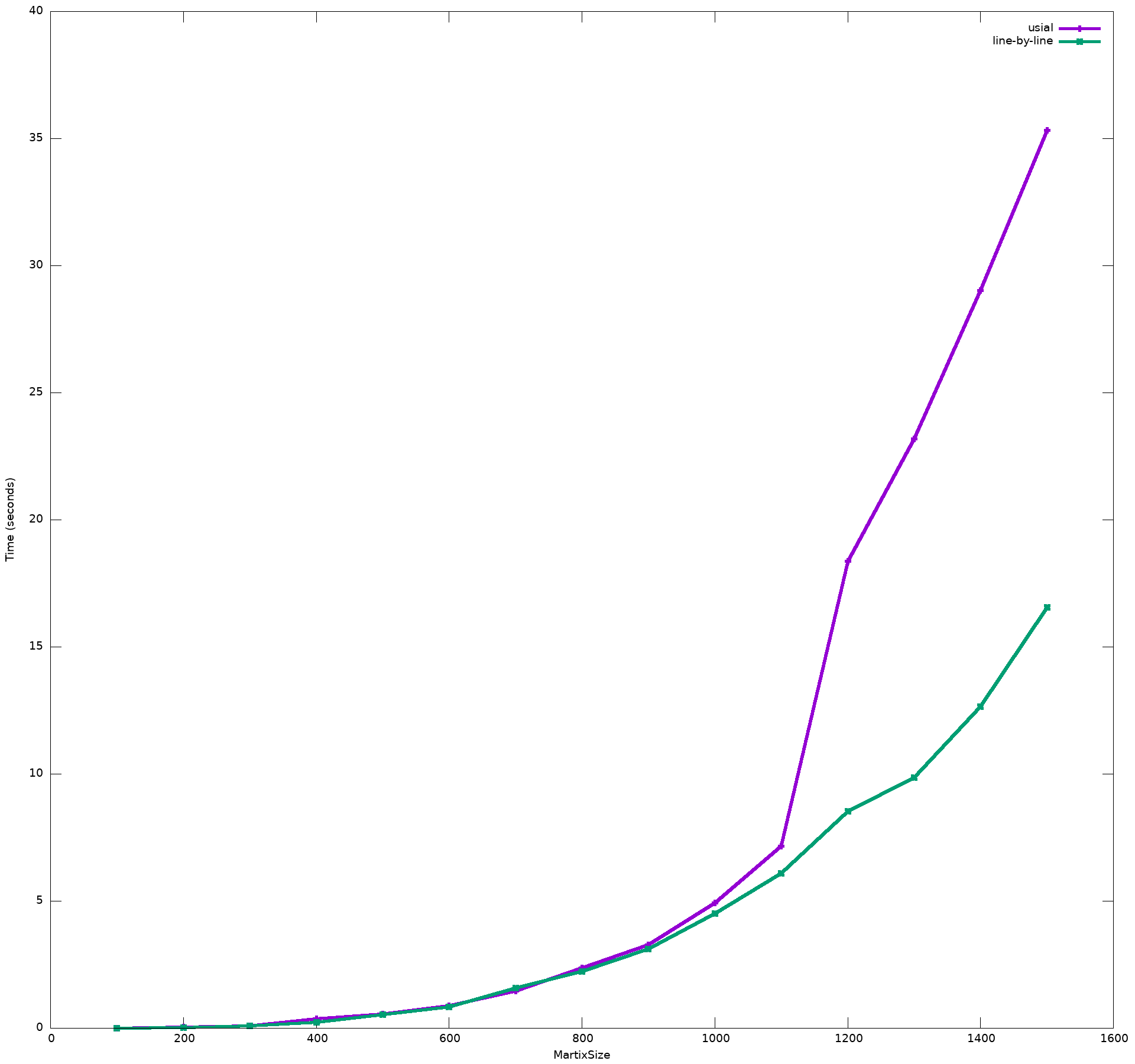
доступа к памяти оценить количество промахов при работе к КЭШ памятью (cache-misses).

1. Подготовить отчёт отражающий суть, этапы и результаты проделанной работы.

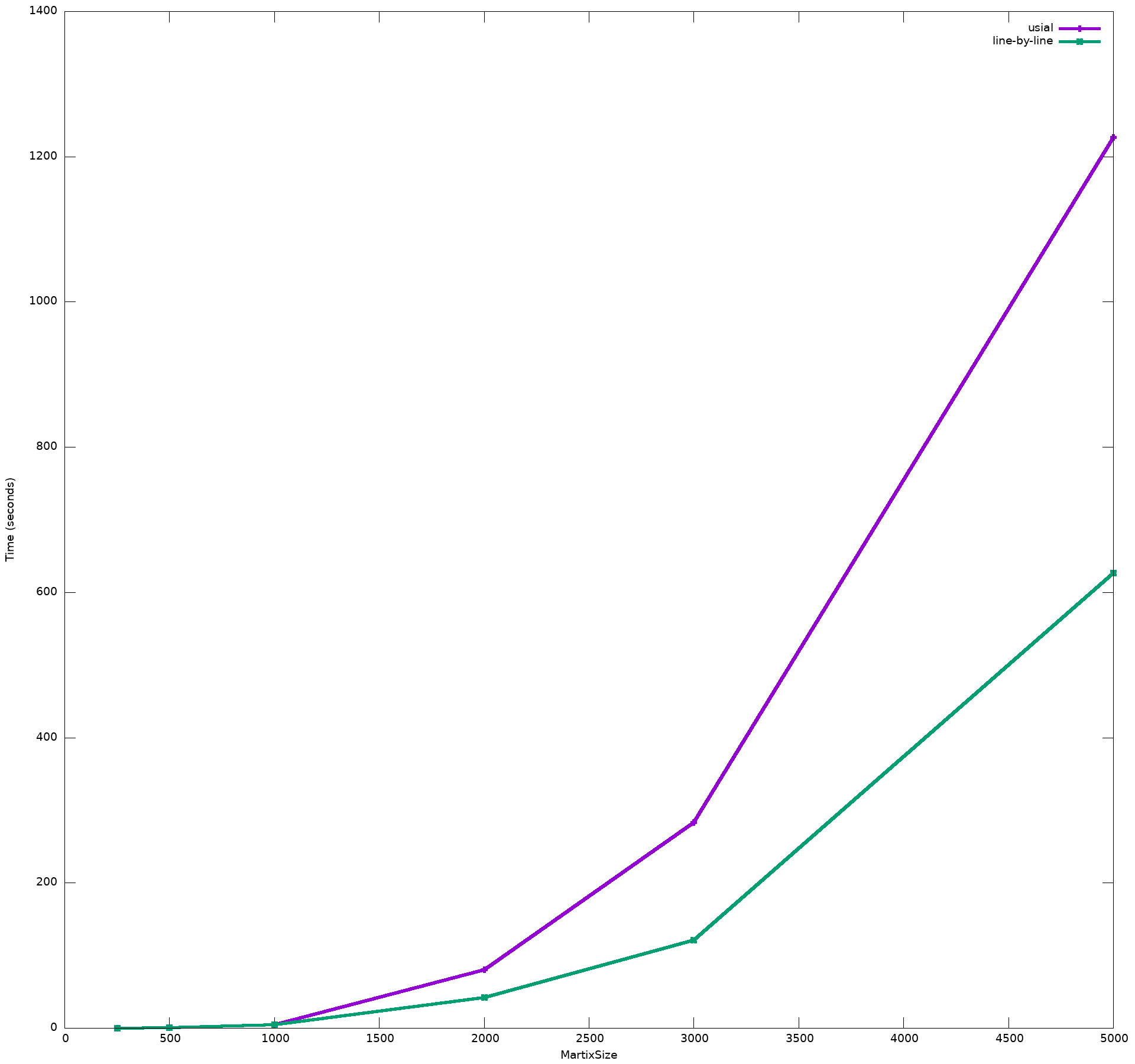
# Примеры работы программы



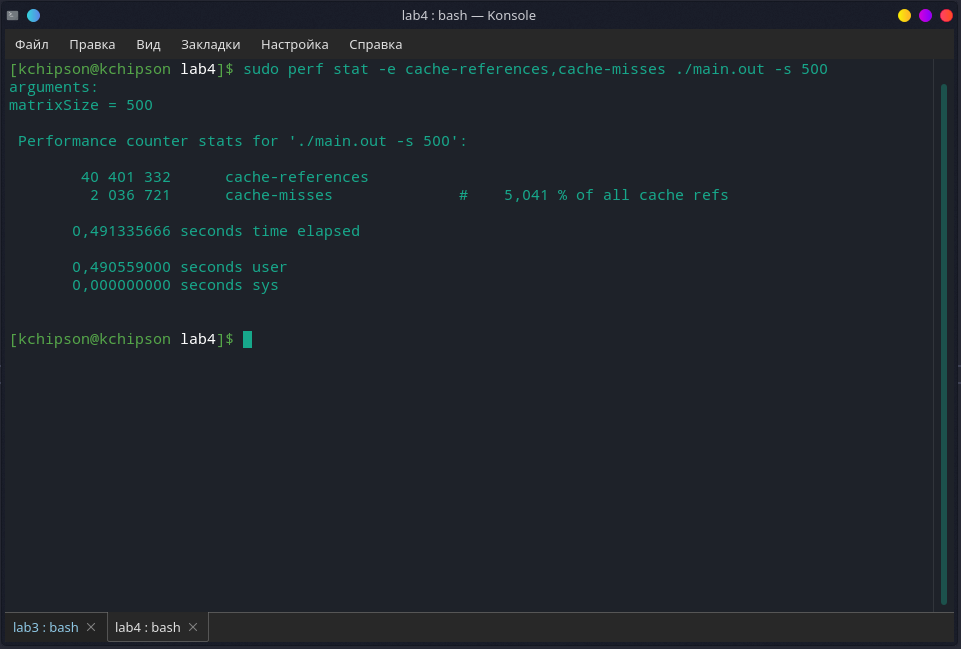
(Часть выходного csv файла)



(График зависимости времени исполнения от размера матриц)



(График зависимости времени исполнения от размера матриц)



(Работа профилировщика)

Таблица кэш-промахов:

|  |  |  |
| --- | --- | --- |
| Размер матрицы | DGEMM\_BLAS  (usual) | DGEMMw\_opt\_1  (line by line) |
| 500\*500 | 2 036 721 | 1 114 032 |
| 1000\*1000 | 11 365 793 | 14 060 729 |

Предельные размеры матриц, которые можно перемножить на данном вычислительном устройстве:

Кол-во доступной ОП – 6,8 ГБ

Кол-во занятой ОП – 1,1 ГБ

6,8ГБ - 1,1ГБ = 5,7 ГБ = 5 836,8 МБ = 5 976 883,2 КБ = 6 120 328 396 Б доступно

6 120 328 396 / 8 = 765 041 049 чисел типа double

В памяти необходимо хранить 3 матрицы, следовательно:

765 041 049 / 3 = 255 013 683

Sqrt(255 013 683) =15 969

# *Приложение* Листинг

**main.cpp**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <ctype.h>

#include <string.h>

#include <stdint.h>

int getParameters(int argc, char \*argv[], long long &matrixSize){

for (int i = 1 ; i < argc ; i++)

{

if (strcmp("-s", argv[i]) == 0 || strcmp("--matrix-size", argv[i]) == 0)

{

i++;

size\_t len = strlen(argv[i]);

for (size\_t j = 0 ; j < len; j++)

if (!isdigit(argv[i][j])){

printf("Error in arguments: invalid value for --matrix-size!\nThe value must be a number!\n");

return 1;

}

matrixSize = atoll(argv[i]);

}

else {

printf("Error in arguments: unknown key \"%s\"\n", argv[i]);

return 1;

}

}

return 0;

}

void printMatrix(double \*\*matrix, long long size){

for (long long i = 0; i < size; i++) {

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

printf("%.6f ", matrix[i][j]);

printf("\n");

}

}

void DGEMM(double\*\* matrixA, double\*\* matrixB, double\*\* matrixC, long long matrixSize){

for (long long i = 0; i < matrixSize; i++)

for (long long j = 0; j < matrixSize; j++)

for (long long k = 0; k < matrixSize; k++)

matrixC[i][j] += matrixA[i][k] \* matrixB[k][j];

}

void DGEMM\_opt1(double\*\* matrixA, double\*\* matrixB, double\*\* matrixC, long long matrixSize){

for (long long i = 0; i < matrixSize; i++)

for (long long k = 0; k < matrixSize; k++)

for (long long j = 0; j < matrixSize; j++)

matrixC[i][j] += (double)matrixA[i][k] \* matrixB[k][j];

}

int outToCSV(char\* type, long long matrixSize, double Time){

FILE \*fp;

if (!(fp = fopen("output.csv", "a"))){

printf("Error: can't open/find output.csv\n");

return 1;

}

fprintf(fp, "%s;%lld;%e;%s;\n", type, matrixSize, Time, "clock()");

fclose(fp);

return 0;

}

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

srand(time(0));

long long matrixSize = 10;

clock\_t start, stop;

if (getParameters(argc, argv, matrixSize))

return 1;

printf("arguments:\n");

printf("matrixSize = %lld\n", matrixSize);

double \*\*matrixA = new double\*[matrixSize];

double \*\*matrixB = new double\*[matrixSize];

double \*\*matrixRes = new double\*[matrixSize];

for (long long i = 0; i < matrixSize; i++) {

matrixA[i] = new double[matrixSize];

matrixB[i] = new double[matrixSize];

matrixRes[i] = new double[matrixSize];

for (long long j = 0; j < matrixSize; j++) {

matrixA[i][j] = rand() / 10000 + (double)rand() / RAND\_MAX;

matrixB[i][j] = rand() / 10000 + (double)rand() / RAND\_MAX;

matrixRes[i][j] = 0;

}

}

double time;

start = clock();

DGEMM(matrixA, matrixB, matrixRes, matrixSize);

stop = clock();

time = ((double)(stop - start)) / CLOCKS\_PER\_SEC;

outToCSV((char\*)"usual", matrixSize, time);

for (long long i = 0; i < matrixSize; i++)

for (long long j = 0; j < matrixSize; j++)

matrixRes[i][j] = 0;

start = clock();

DGEMM\_opt1(matrixA, matrixB, matrixRes, matrixSize);

stop = clock();

time = ((double)(stop - start)) / CLOCKS\_PER\_SEC;

outToCSV((char\*)"line-by-line", matrixSize, time);

for (long long i = 0; i < matrixSize; i++) {

delete(matrixA[i]);

delete(matrixB[i]);

delete(matrixRes[i]);

}

delete[](matrixA);

delete[](matrixB);

delete[](matrixRes);

return 0;

}

// Кэш-промахи:

// sudo perf stat -e cache-references,cache-misses ./main.out -s 500