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.

Новосибирск 2020-21 уч. год

Оглавление

1. Постановка задачи

Тема: оптимизация доступа к памяти.
Задание:

1. На языке C/C++/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.

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, 1000x1000, 2000x2000, 5000x5000, 10000x10000.
   * Для блочного умножения матриц определить размер блока, при котором достигается максимальное ускорение.

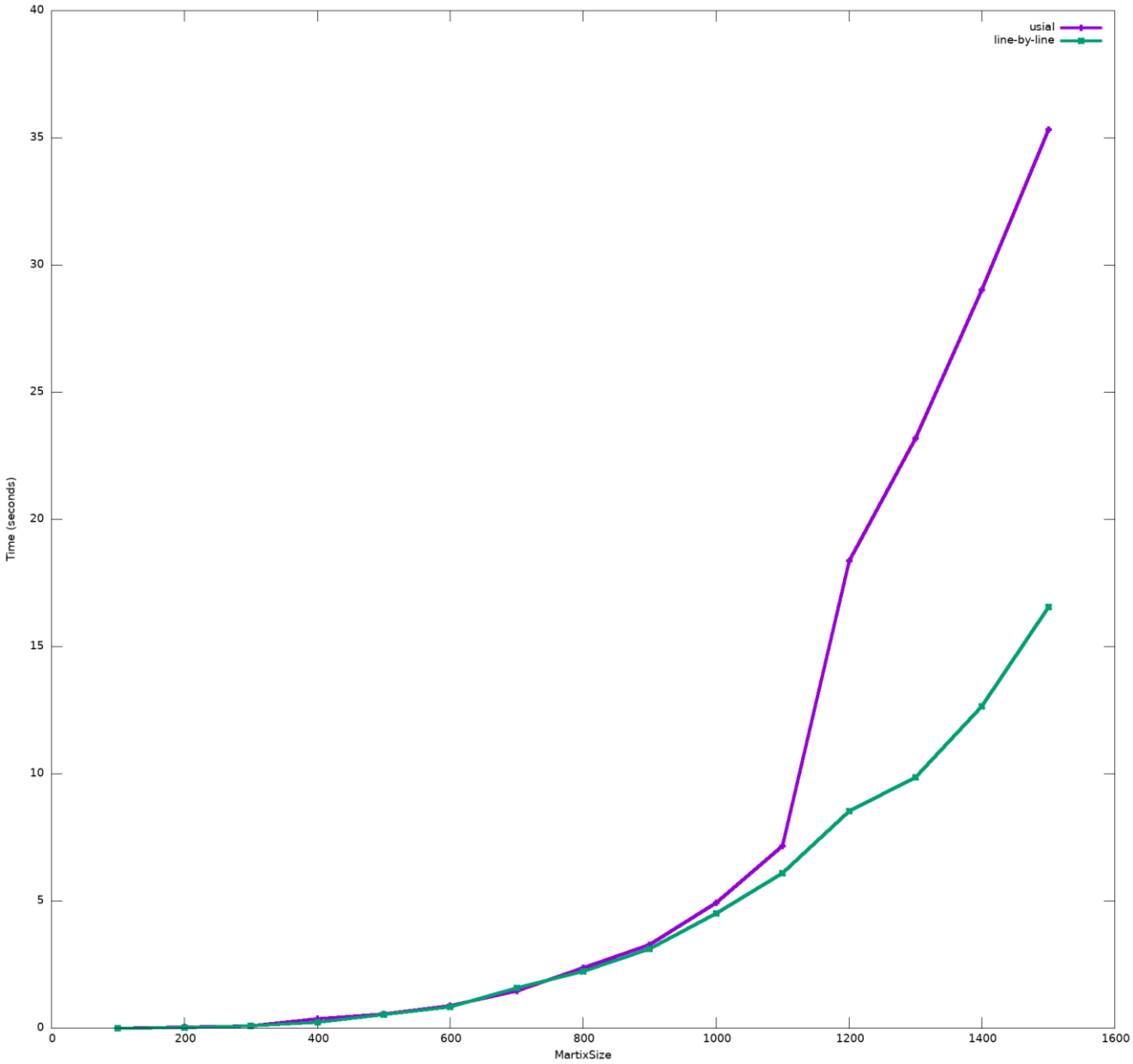
8. С помощью профилировщика для исходной программы и каждого способа оптимизации
   доступа к памяти оценить количество промахов при работе к КЭШ памятью (cache-misses).

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

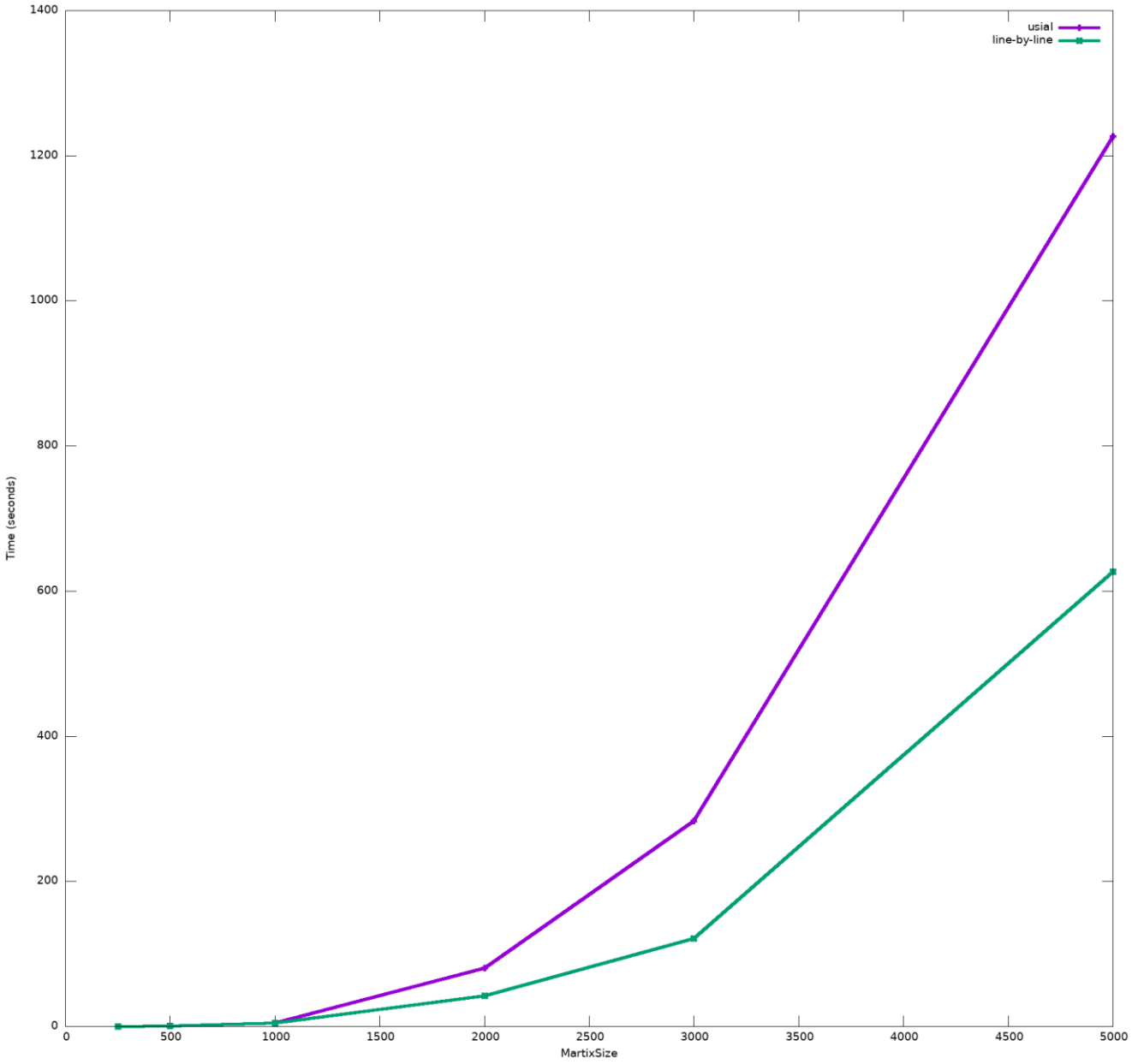
2. Примеры работы программы

	1	2	3	4	5	6
1	Type	MatrixSize	Time	Timer		
2	usual	100	4.925000e-03	clock()		
3	line-by-line	100	3.736000e-03	clock()		
4	usual	200	5.257300e-02	clock()		
5	line-by-line	200	2.965100e-02	clock()		
6	usual	300	1.006380e-01	clock()		
7	line-by-line	300	9.988900e-02	clock()		
8	usual	400	3.636240e-01	clock()		
9	line-by-line	400	2.387650e-01	clock()		
10	usual	500	5.477800e-01	clock()		
11	line-by-line	500	5.442800e-01	clock()		
12	usual	600	8.754560e-01	clock()		
13	line-by-line	600	8.331450e-01	clock()		
14	usual	700	1.459025e+00	clock()		
15	line-by-line	700	1.581416e+00	clock()		
16	usual	800	2.366213e+00	clock()		
17	line-by-line	800	2.226976e+00	clock()		
18	usual	900	3.274740e+00	clock()		
19	line-by-line	900	3.113325e+00	clock()		
20	usual	1000	4.925843e+00	clock()		
21	line-by-line	1000	4.516031e+00	clock()		
22	usual	1100	7.169724e+00	clock()		
23	line-by-line	1100	6.100271e+00	clock()		
24	usual	1200	1.837968e+01	clock()		
25	line-by-line	1200	8.540340e+00	clock()		
26	usual	1300	2.317936e+01	clock()		
27	line-by-line	1300	9.865128e+00	clock()		
28	usual	1400	2.903057e+01	clock()		
29	line-by-line	1400	1.264134e+01	clock()		
30	usual	1500	3.533058e+01	clock()		
31	line-by-line	1500	1.655130e+01	clock()		

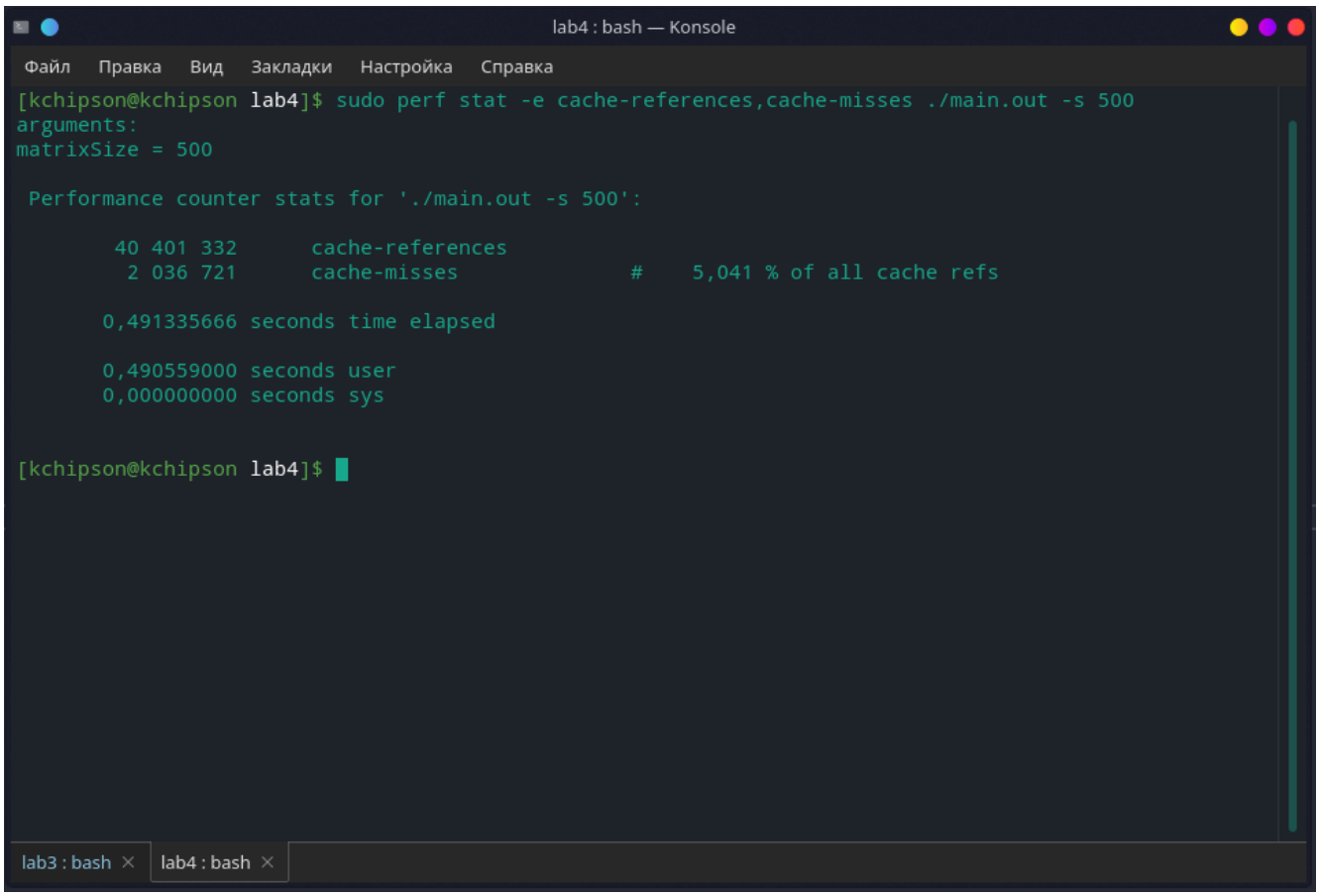
(Часть выходного csv файла)



(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
```