ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КАФЕДРА ВС

Отчёт по лабораторной работе №4

«Оптимизация доступа к памяти.»

по дисциплине «Архитектура вычислительных систем»

Выполнил: студент гр. ИВ-823
Шиндель Э.Д.
Проверил: ст. преп. Кафедры ВС
Токмашева Е.И.

Новосибирск 2020

Содержание

Постановка задачи

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. Оценить ускорение умножения для матриц фиксированного размера, например, 1000x1000, 2000x2000, 5000x5000, 10000x10000.

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

Результат работы

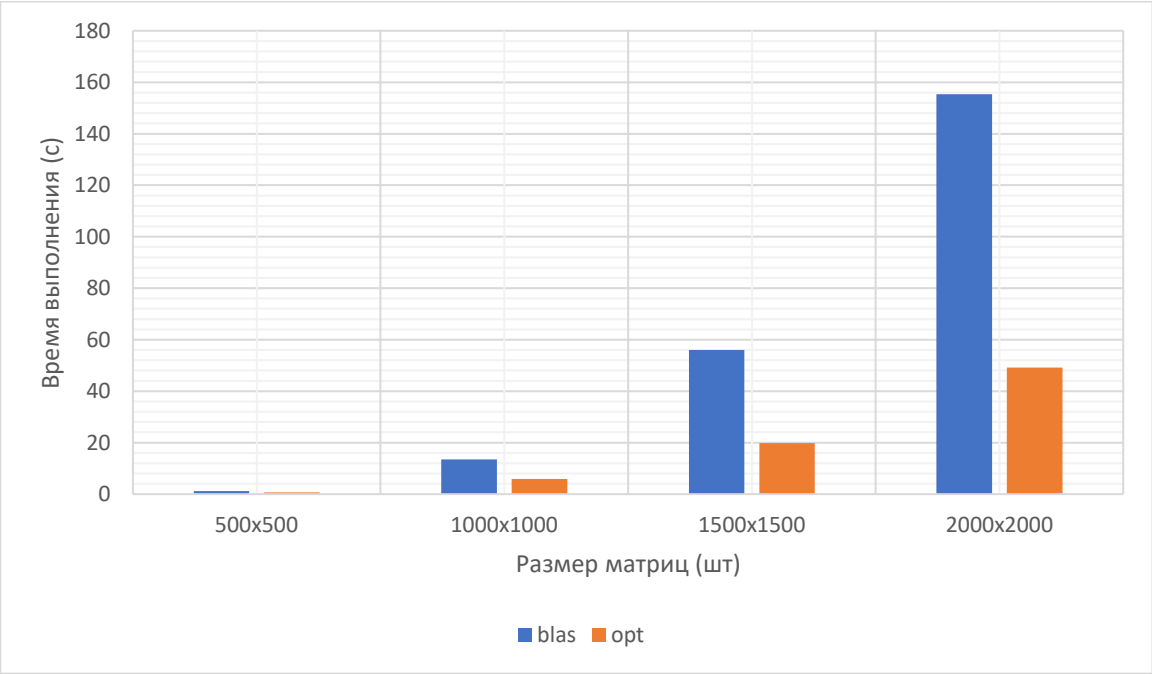

График зависимости времени выполнения программы от объёма входных данных.

N	Dgemm_blas	Dgemm_opt	SpeedUp
500	1,21	0,73	1,65
1000	13,46	5,88	2,29
1500	56	19,77	2,83
2000	155,36	49,16	3,16

У меня 4 GB оперативной памяти. Матрица квадратная (n = m). В нашей программе хранятся 3 матрицы (n · n) ячеек, каждая ячейка типа double занимает 8 байт. Тогда под массивы надо 8 · (3 · n · n) байт памяти. На практике занимать всю память нельзя, возьмём 80% от 4 GB – это 0,8 · 4 GB. Составим уравнение и найдём решение: $8 \cdot (3 \cdot n \cdot n) = 0{,}8 \cdot 4 \cdot 2^{30}$. Получилось, что предельные размеры матрицы, которые можно перемножить на моём вычислительном устройстве ≈ 11965.

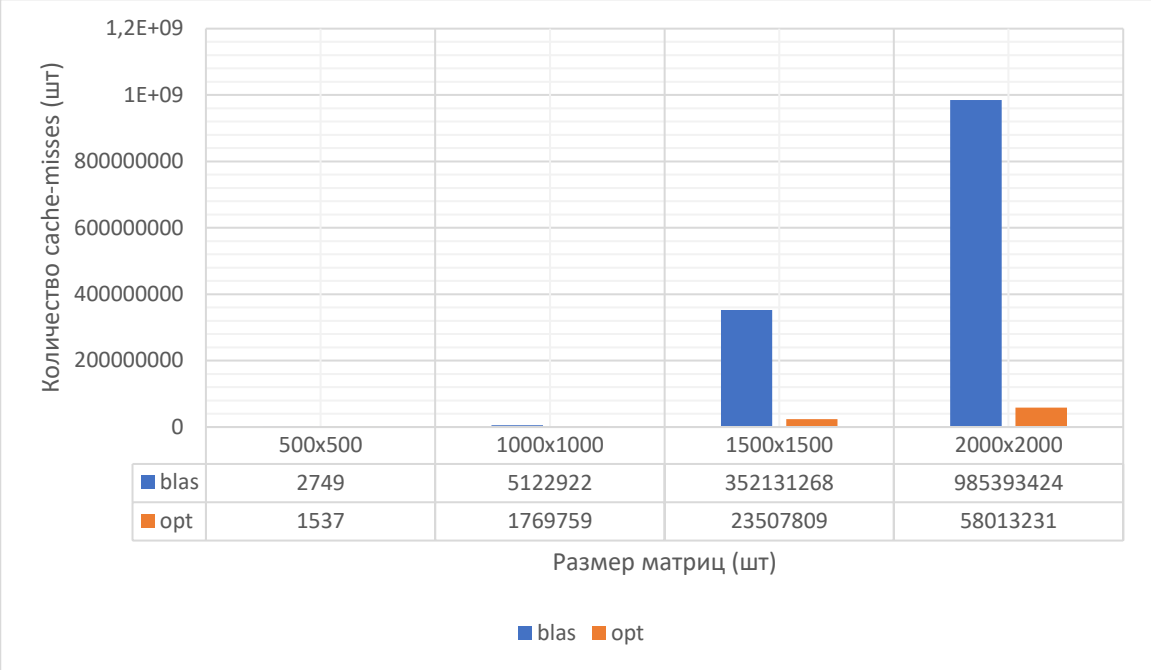

График зависимости cache-misses в программе от объёма входных данных.

Приложение

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>

double wtime()
{
    struct timeval t;
    gettimeofday(&t, NULL);
    return (double)t.tv_sec + (double)t.tv_usec * 1E-6;
}

void dgemm_blas(double **A, double **B, double **C, int size)
{
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            for (int k = 0; k < size; k++) {
                C[i][j] += A[k][j] * B[i][k];
            }
        }
    }
}

void dgemm_opt(double **A, double **B, double **C, int size)
{
    for (int i = 0; i < size; i++) {
        for (int k = 0; k < size; k++) {
            for (int j = 0; j < size; j++) {
                C[i][j] += A[k][j] * B[i][k];
            }
        }
    }
}

int main (int argc, char *argv[])
{
    if (argc != 2) {
        printf("ERROR!\n");
        return 1;
    }
    int size = atoi(argv[1]);
    srand(time(NULL));

    double **A = (double **) malloc(sizeof(double *) * size);
    double **B = (double **) malloc(sizeof(double *) * size);
    double **C = (double **) malloc(sizeof(double *) * size);
    for (int i = 0; i < size; i++) {
        A[i] = (double *) malloc(sizeof(double) * size);
        B[i] = (double *) malloc(sizeof(double) * size);
        C[i] = (double *) malloc(sizeof(double) * size);
        for (int j = 0; j < size; j++) {
            A[i][j] = (double) (rand() % 201 - 100);
            B[i][j] = (double) (rand() % 201 - 100);
            C[i][j] = 0.0;
        }
    }

    double time_1 = wtime();
    dgemm_blas(A, B, C, size);
    time_1 = wtime() - time_1;
```

```
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            A[i][j] = (double) (rand() % 201 - 100);
            B[i][j] = (double) (rand() % 201 - 100);
            C[i][j] = 0.0;
        }
    }

    double time_2 = wtime();
    dgemm_opt(A, B, C, size);
    time_2 = wtime() - time_2;

    printf("Time(dgemm_blas) = %.2f sec\nTime(dgemm_opt) = %.2f sec\nSpeedup = %.2f\n",
                                                                    time_1, time_2, time_1 / time_2);
    free(A);
    free(B);
    free(C);
    return 0;
}
```