

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. ИП-811

Разумов Д.Б.

Проверил: ст. преп. Кафедры ВС

Ткачёва Т.А.

Новосибирск 2020

Содержание

Постановка задачи.....	3
Выполнение работы.....	4
Задание 1.....	4
Задание 2.....	5
Задание 3.....	6
Задания 4 и 5.....	7
Задание 6.....	7
Задание 6*.....	8
Задание 7.....	9
Приложение.....	12
Файл source/main.cpp.....	12
Файл source/foo.h.....	12
Файл source/foo.cpp.....	13
Файл scripts/s2.sh.....	18
Файл scripts/d2.gpi.....	18
Файл scripts/s6.sh.....	19
Файл scripts/d6.gpi.....	19
Файл scripts/star1.sh.....	20
Файл scripts/star1.gpi.....	20

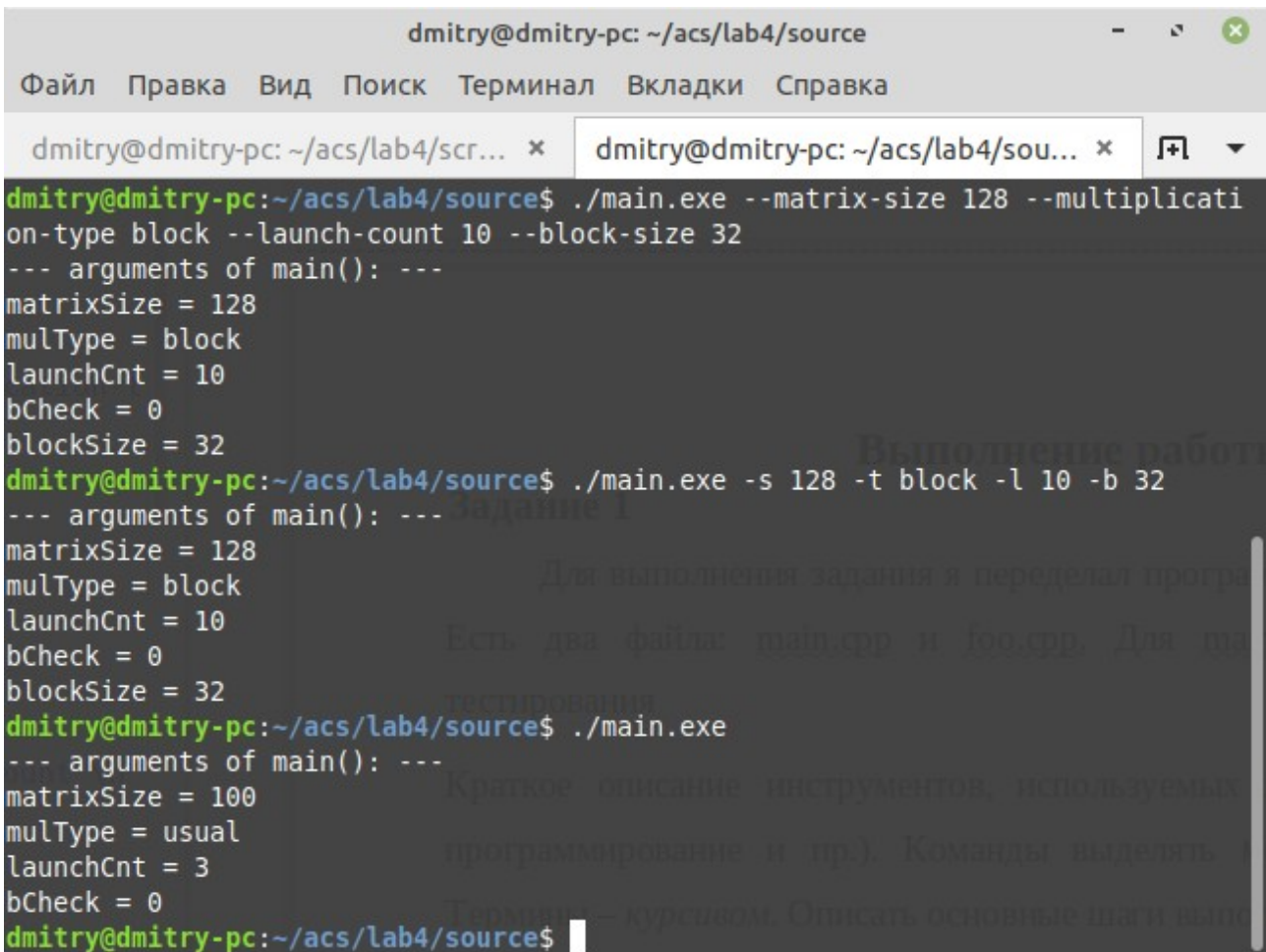
Постановка задачи

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 персональном компьютере (ПК).
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. Оценить ускорение умножения для матриц фиксированного размера, например, 1000x1000, 2000x2000, 5000x5000, 10000x10000.
* Для блочного умножения матриц определить размер блока, при котором достигается максимальное ускорение.
7. С помощью профилировщика для исходной программы и каждого способа оптимизации доступа к памяти оценить количество промахов при работе к КЭШ памятью (cache -misses).
8. Подготовить отчет отражающий суть, этапы и результаты проделанной работы.

Выполнение работы

Задание 1

Для выполнения задания я переделал программу из предыдущей работы. Есть два файла: main.cpp и foo.cpp. Для main.cpp задаются параметры тестирования вместе с исполнением main.exe.



```
dmitry@dmitry-pc: ~/acs/lab4/source
Файл  Правка  Вид  Поиск  Терминал  Вкладки  Справка
dmitry@dmitry-pc: ~/acs/lab4/scr... x  dmitry@dmitry-pc: ~/acs/lab4/sou... x
dmitry@dmitry-pc:~/acs/lab4/source$ ./main.exe --matrix-size 128 --multiplicati
on-type block --launch-count 10 --block-size 32
--- arguments of main(): ---
matrixSize = 128
mulType = block
launchCnt = 10
bCheck = 0
blockSize = 32
dmitry@dmitry-pc:~/acs/lab4/source$ ./main.exe -s 128 -t block -l 10 -b 32
--- arguments of main(): ---
matrixSize = 128
mulType = block
launchCnt = 10
bCheck = 0
blockSize = 32
dmitry@dmitry-pc:~/acs/lab4/source$ ./main.exe
--- arguments of main(): ---
matrixSize = 100
mulType = usual
launchCnt = 3
bCheck = 0
dmitry@dmitry-pc:~/acs/lab4/source$
```

Изображение 1: примеры запуска программы

В main.cpp вызываются следующие функции: ProcessParameters() для обработки входных параметров и TestsHandler() для запуска нужного тестирования. Также здесь задаются параметры по умолчанию, поэтому пользователю необязательно вводить все параметры.

В foo.cpp есть следующие функции:

- ProcessParameters(), упомянутая выше;

- `GetCacheAlignment()` для нахождения выравнивания кэша, которое используется по умолчанию для размера блока в блочном умножении матриц;
- `MatrixMul()` - в этой функции происходит обычное и построчное умножение матриц с заданными параметрами;
- `MatrixBlockMul()` - в этой функции происходит блочное умножение матриц с заданными параметрами;
- `WriteToCSV()` для записи результатов тестирования в файл `output.csv`;
- `TestsHandler()` для запуска нужного умножения `launch-count` раз и вычисления времени и погрешностей.

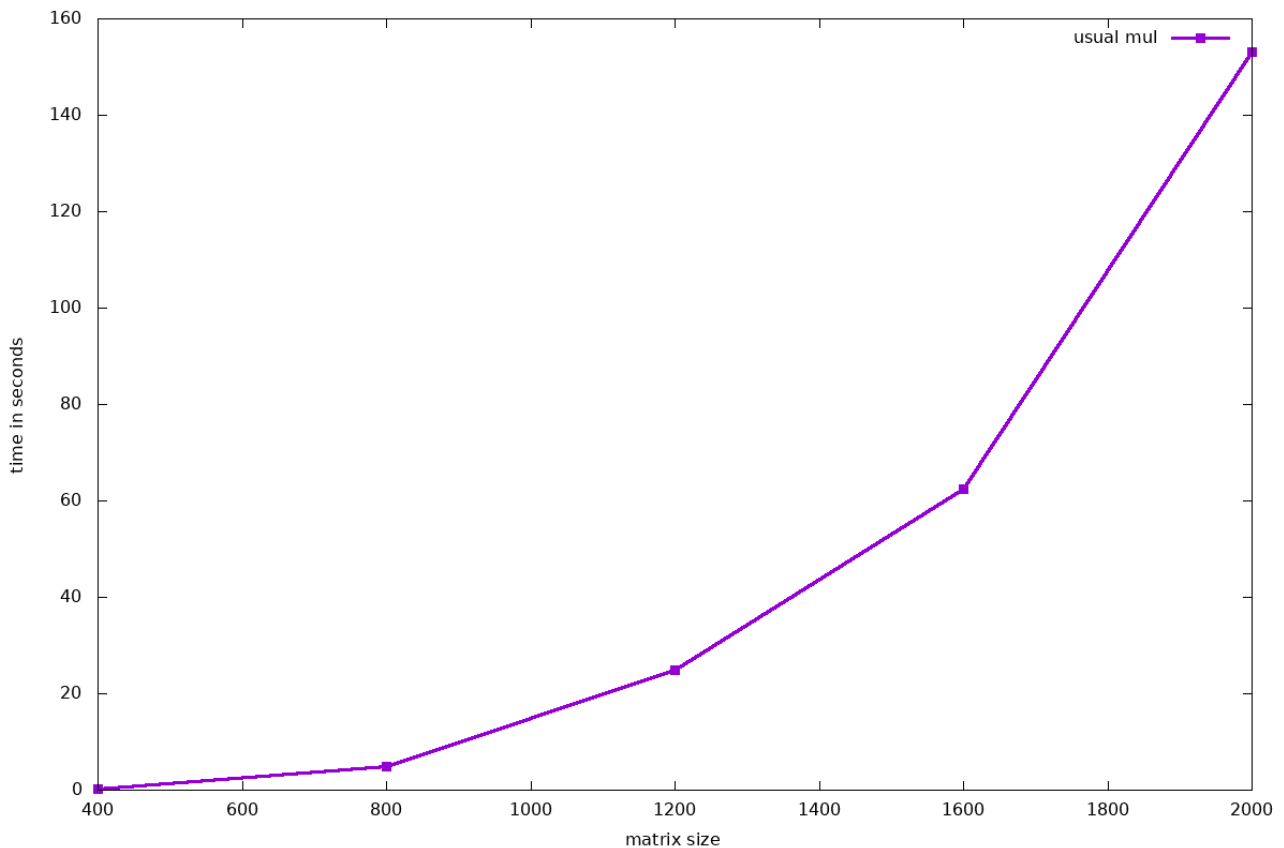
Также несколько необязательных (для выполнения лабораторной) функций: `min()` и `PrintMatrix()`.

Задание 2

Тестирование осуществляется с помощью Bash- и gnuplot-скриптов. В данном задании в Bash-скрипте программа из задания 1 запускается несколько раз в цикле. Размерность матрицы меняется от 400 до 2000 с шагом 400.

	A	B	C	D	E	F	G	H
1	<u>multiplication type</u>	<u>launch count</u>	<u>matrix size</u>	<u>block size</u>	<u>timer</u>	<u>average time</u>	<u>absError</u>	<u>relError</u>
2	<u>usual</u>	3	400	64	<u>clock()</u>	3.313057e-01	1.355682e-02	5.547367e-02%
3	<u>usual</u>	3	800	64	<u>clock()</u>	4.980163e+00	5.413937e-01	5.885492e+00%
4	<u>usual</u>	3	1200	64	<u>clock()</u>	2.488900e+01	7.366408e-01	2.180239e+00%
5	<u>usual</u>	3	1600	64	<u>clock()</u>	6.250155e+01	9.445804e-01	1.427536e+00%
6	<u>usual</u>	3	2000	64	<u>clock()</u>	1.530029e+02	8.309214e+00	4.512533e+01%

Изображение 2: содержимое `output.csv` после выполнения скрипта (задание 2)



Изображение 3: график, построенный на данных output.csv (задание 2)

Примечание: максимальная размерность 2000, так как в теории в одном запуске MatrixMul() для matrix-size=4000 примерно $4000 \cdot 4000 \cdot 4 + 4000^3 \cdot 2 = 128 \cdot 10^9$ операций, среди которых rand() и вещественное умножение и деление. На практике при matrix-size=4000, программа выполняется больше 10 минут на моем компьютере.

Задание 3

Сразу после запуска операционной системы на моем компьютере свободно 2,6 Гб (узнал с помощью команды `free -h`). В функциях MatrixMul() и MatrixBlockMul создаются три динамических массива одинаковых по размерности. Количество строк и столбцов всегда равно по условию. Тип данных массивов — double, который весит 8 байт (проверил с помощью `sizeof(double)`). Другими переменными можно пренебречь, так как они в сравнении очень малы.

1. Сначала нужно перевести Гб в байты и поделить на размер double:

$$2.6 * 1024 * 1024 * 1024 / 8 = 348\,966\,092$$

2. Полученное число поделить на 3, так как матриц 3:

$$348\,966\,092 / 3 = 116\,322\,030$$

3. Взять корень из полученное числа, так как массив двумерный:

$$\text{sqrt}(116\,322\,030) = 10\,785.$$

Ответ: на моем компьютере можно перемножить матрицы с максимальной размерностью 10785 на 10785.

Задания 4 и 5

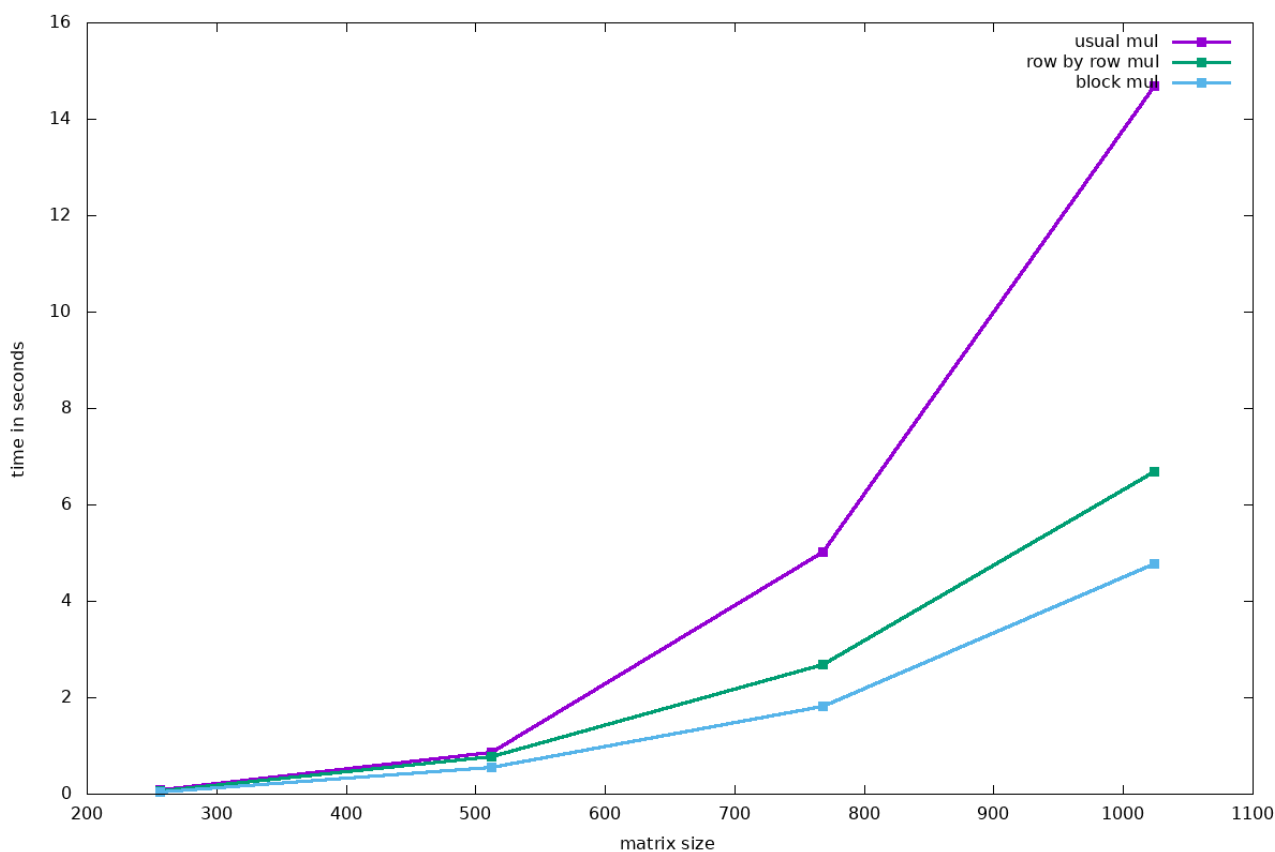
Построчное и блочное умножение матриц реализовано в точности как в справочном материале.

Задание 6

Это задание выполнено аналогично заданию 2 с помощью скриптов.

	A	B	C	D	E	F	G	H
1	multiplication type	launch count	matrix size	block size	timer	average time	absError	relError
2	usual	3	256	64	clock()	8.692633e-02	6.303710e-03	4.571315e-02%
3	row_by_row	3	256	64	clock()	6.370933e-02	1.547217e-03	3.757503e-03%
4	block	3	256	64	clock()	4.326267e-02	1.480413e-04	5.065851e-05%
5	usual	3	512	64	clock()	8.640923e-01	1.763270e-01	3.598135e+00%
6	row_by_row	3	512	64	clock()	7.739453e-01	1.287007e-03	2.140186e-04%
7	block	3	512	64	clock()	5.591537e-01	3.001203e-02	1.610867e-01%
8	usual	3	768	64	clock()	5.027671e+00	1.474517e-02	4.324467e-03%
9	row_by_row	3	768	64	clock()	2.695683e+00	2.810600e-02	2.930416e-02%
10	block	3	768	64	clock()	1.828215e+00	2.523554e-02	3.483356e-02%
11	usual	3	1024	64	clock()	1.468620e+01	2.443557e-01	4.065703e-01%
12	row_by_row	3	1024	64	clock()	6.690677e+00	5.360297e-01	4.294451e+00%
13	block	3	1024	64	clock()	4.780547e+00	1.320106e-01	3.645355e-01%

Изображение 4: output.csv (задание 6)



Изображение 5: график, построенный на данных output.csv (задание 6)

Оценка ускорения при matrix-size = 1024:

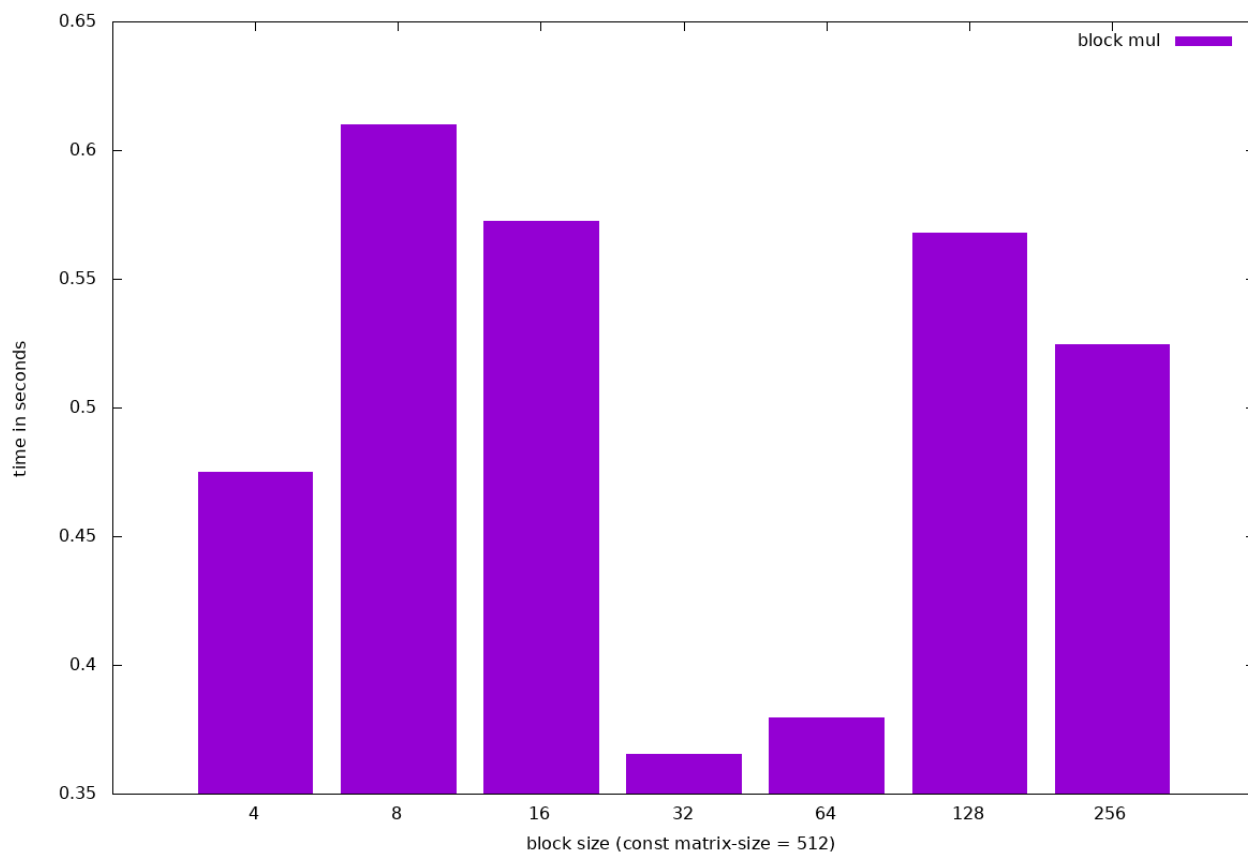
	Обычное умножение	Построчное умножение	Блочное умножение
Время в секундах	14.686	6.6961	4.7805
Ускорение	-	2.1932	3.0721

Задание 6*

Здесь в скрипте размер матрицы не меняется для всех тестов и равен 512. Но меняется block-size от 4 до 256. Как видно на диаграмме, самое меньшее время выполнения у block-size = 32. Следовательно, при заданных условиях максимальное ускорение достигается при размере блока 32 на 32 элемента.

	A	B	C	D	E	F	G	H
1	<u>multiplication type</u>	<u>launch count</u>	<u>matrix size</u>	<u>block size</u>	<u>timer</u>	<u>average time</u>	<u>absError</u>	<u>relError</u>
2	block	3	512	4	clock()	4.748500e-01	3.460643e-03	2.522070e-03%
3	block	3	512	8	clock()	6.101677e-01	5.775120e-02	5.466039e-01%
4	block	3	512	16	clock()	5.725150e-01	2.477485e-02	1.072100e-01%
5	block	3	512	32	clock()	3.654990e-01	1.899675e-02	9.873532e-02%
6	block	3	512	64	clock()	3.794253e-01	3.938397e-02	4.088018e-01%
7	block	3	512	128	clock()	5.677887e-01	3.390137e-02	2.024174e-01%
8	block	3	512	256	clock()	5.245490e-01	3.869005e-02	2.853728e-01%

Изображение 6: output.csv (задание 6*)



Изображение 7: диаграмма, построенная на данных output.csv (задание 6*)

Задание 7

При matrix-size = 512 и launch-count = 3 количество кэш-промахов для обычного, построчного и блочного умножения равно $6 \cdot 10^6$, $23 \cdot 10^6$ и $3 \cdot 10^6$ соответственно.

```

dmitry@dmitry-pc:~/acs/lab4/source$ sudo perf stat -e cache-references,cache-misses ./main.exe
-s 512 -t usual
[sudo] пароль для dmitry:
--- arguments of main(): ---
matrixSize = 512
mulType = usual
launchCnt = 3
bCheck = 0

Лабораторная работа № 4. Оптимизация доступа к памяти.

Performance counter stats for './main.exe -s 512 -t usual':
      82 489 968      cache-references
       6 316 609      cache-misses # 7,657 % of all cache refs
      2,272327492 seconds time elapsed

      2,245754000 seconds user
      0,008006000 seconds sys

3. Оценить предельные размеры матриц, которые можно переместить в кэш процессора.
4. Реализовать дополнительную функцию DGEMM_opt_1, в которой оптимизация доступа к памяти, за счет строчного перебора элементов матрицы.
5. * Реализовать дополнительную функцию DGEMM_opt_2, в которой оптимизация доступа к памяти, за счет блочного перебора элементов матрицы.
6. Оценить ускорение умножения для матриц фиксированного размера.
7. С помощью профилирования для исходной программы и каждой из оптимизированных программ оценить количество промахов при работе с кэшем.
8. Подготовить отчет отражающий суть, эталон и результаты проведенных экспериментов.

dmitry@dmitry-pc:~/acs/lab4/source$ sudo perf stat -e cache-references,cache-misses ./main.exe
-s 512 -t row_by_row
--- arguments of main(): ---
matrixSize = 512
mulType = row_by_row
launchCnt = 3
bCheck = 0

Performance counter stats for './main.exe -s 512 -t row_by_row':
      126 298 513      cache-references
       23 694 705      cache-misses # 18,761 % of all cache refs
      1,625217806 seconds time elapsed

      1,606151000 seconds user
      0,000000000 seconds sys

```

Изображение 8: кэш-промахи для обычного и строчного умножения

```

dmitry@dmitry-pc:~/acs/lab4/source$ sudo perf stat -e cache-references,cache-misses ./main.exe
-s 512 -t block
--- arguments of main(): ---7. С помощью профилировщика для исходной программы и каждого сп
matrixSize = 512
mulType = block
launchCnt = 3
bCheck = 0
blockSize = 64
8. Подготовить отчет отражающий суть, этапы и результаты проделанно

Материалы для самостоятельного изучения:
1. Лекция 3 «Оптимизация доступа к памяти» (М.Г. Курносов) https://mkurnosov.github.io/lectures/3/

Performance counter stats for './main.exe -s 512 -t block':

    79 530 065      cache-references
    3 554 903      cache-misses          #    4,470 % of all cache refs

    1,163374030 seconds time elapsed

    1,136622000 seconds user
    0,008032000 seconds sys

```

Изображение 9: кэш-промахи для блочного умножения

Приложение

Файл source/main.cpp

```
#include "foo.h"

int main(int argc, char *argv[]) {
    srand(time(0));
    //printf("sizeof(double)=%d \n", sizeof(double));
    ///инициализация параметров:
    long long matrixSize = 100;
    char* mulType = (char*) malloc(15);
    strcpy(mulType, "usual\0");
    long long launchCnt = 3;
    bool bCheck = false;
    long long blockSize = GetCacheAlignment();
    ProcessParameters(argc, argv, matrixSize, mulType, launchCnt, bCheck,
blockSize);
    ///проверка параметров:
    printf("--- arguments of main(): --- \n");
    printf("matrixSize = %lld \n", matrixSize);
    printf("mulType = %s \n", mulType);
    printf("launchCnt = %lld \n", launchCnt);
    printf("bCheck = %d \n", bCheck ? 1 : 0);
    if (strcmp("block", mulType) == 0)
        printf("blockSize = %lld \n", blockSize);
    ///запуск:
    TestsHandler(mulType, launchCnt, matrixSize, bCheck, blockSize);

    free(mulType);
    return 0;
}
```

Файл source/foo.h

```
#ifndef F00
#define F00

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <stdint.h> //подключение фиксированных типов данных

//int UsualMatrixMul(long long n, double &time_d);
int ProcessParameters(int argc, char *argv[],
                    long long &matrixSize, char* mulType,
                    long long &launchCnt, bool &bCheck,
                    long long &blockSize);
int TestsHandler(char* mulType, long long launchCnt,
                long long matrixSize, bool bCheck,
                long long blockSize);
long long GetCacheAlignment();

#endif
```

Файл source/foo.cpp

```
#include "foo.h"

int ProcessParameters(int argc, char *argv[],
                     long long &matrixSize, char* mulType,
                     long long &launchCnt, bool &bCheck,
                     long long &blockSize)
{
    int i;
    for (i = 1; i < argc; i++)
    {
        ///если размер матрицы:
        if (strcmp("-s", argv[i]) == 0 ||
            strcmp("--matrix-size", argv[i]) == 0)
        {
            i++;
            matrixSize = atoll(argv[i]); ///приведение строки в long long int
            if (matrixSize == 0) {
                printf("Error in arguments of main(): incorrect value for --
launch-count \n");
                return 1;
            }
        }
        ///если тип умножения:
        else if (strcmp("-t", argv[i]) == 0 ||
                 strcmp("--multiplication-type", argv[i]) == 0)
        {
            i++;
            if (strcmp("usual", argv[i]) == 0 ||
                strcmp("row_by_row", argv[i]) == 0 ||
                strcmp("block", argv[i]) == 0)
            {
                strcpy(mulType, argv[i]);
            }
            else {
                printf("Error in arguments of main(): incorrect value for --
multiplication-type \n");
                return 1;
            }
        }
        ///если число испытаний:
        else if (strcmp("-l", argv[i]) == 0 ||
                 strcmp("--launch-count", argv[i]) == 0)
        {
            i++;
            launchCnt = atoll(argv[i]); ///приведение строки в long long int
            if (launchCnt == 0) {
                printf("Error in arguments of main(): incorrect value for --
launch-count \n");
                return 1;
            }
        }
        ///если включение проверки умножения:
        else if (strcmp("-c", argv[i]) == 0 ||
                 strcmp("--check", argv[i]) == 0)
        {
            bCheck = true;
        }
    }
}
```

```

    }
    else if (strcmp("-b", argv[i]) == 0 ||
             strcmp("--block-size", argv[i]) == 0)
    {
        i++;
        blockSize = atoll(argv[i]); ///приведение строки в long long int
        if (blockSize == 0) {
            printf("Error in arguments of main(): incorrect value for --
block-size \n");
            return 1;
        }
    }
}

return 0;
}

long long min(long long a, long long b) {
    if (a < b)
        return a;
    else
        return b;
}

long long GetCacheAlignment() {
    FILE *fcpu;
    if ((fcpu = fopen("/proc/cpuinfo", "r")) == NULL) {
        printf("Error: can't open /proc/cpuinfo \n");
        return -1;
    }
    size_t m = 0;
    char *line = NULL, *temp = (char*) malloc(50);
    while (getline(&line, &m, fcpu) > 0) {
        if (strstr(line, "cache_alignment")) {
            strcpy(temp, &line[18]);
            break;
        }
    }

    for (int i = 0; i < 50; i++) {
        if (temp[i] == ' ' || temp[i] == '\n') {
            temp[i] = '\0';
            //strncpy(temp, temp, i);
            //break;
        }
    }
    //printf("temp=%s\n", temp);
    long long val = atoll(temp);
    if (val == 0) {
        printf("Error in GetCacheAlignment(): can't atoll \n");
        return -1;
    }

    fclose(fcpu);
    free(temp);
    return val;
}

```

```

int PrintMatrix(double **matrix, long long n) {
    long long i, j;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            //printf("[%lld][%lld]=", i, j);
            printf("%.6f ", matrix[i][j]);
        }
        printf("\n");
    }
    return 0;
}

int MatrixMul(int mulType_i, long long matrixSize,
              bool bCheck, double &time_d) {
    long long i, j, k;
    ///выделение памяти под матрицы:
    double **matrix1 = new double*[matrixSize];
    double **matrix2 = new double*[matrixSize];
    double **matrixRes = new double*[matrixSize];
    for (i = 0; i < matrixSize; i++) {
        matrix1[i] = new double[matrixSize];
        matrix2[i] = new double[matrixSize];
        matrixRes[i] = new double[matrixSize];
    }
    ///заполнение матриц случайными числами:
    for (i = 0; i < matrixSize; i++) {
        for (j = 0; j < matrixSize; j++) {
            matrix1[i][j] = rand() / 123456 + (double)rand() / RAND_MAX;
            matrix2[i][j] = rand() / 123456 + (double)rand() / RAND_MAX;
            matrixRes[i][j] = 0;
        }
    }
    ///проверка:
    if (bCheck) {
        printf("before multiplication: \n");
        PrintMatrix(matrix1, min((long long)3, matrixSize));
        printf("\n");
        PrintMatrix(matrix2, min((long long)3, matrixSize));
        printf("\n");
        PrintMatrix(matrixRes, min((long long)3, matrixSize));
        printf("\n");
    }
    ///для вычисления времени:
    clock_t start, stop;
    long long time_i = 0;
    start = clock();
    ///умножение матриц по строке и столбцу:
    if (mulType_i == 1) { ///обычное умножение матриц
        for (i = 0; i < matrixSize; i++)
            for (j = 0; j < matrixSize; j++)
                for (k = 0; k < matrixSize; k++) {
                    ///если замерять время здесь, то программа многократно замедляется
                    matrixRes[i][j] += matrix1[i][k] * matrix2[k][j];
                }
    }
    else if (mulType_i == 2) { ///построчное умножение матриц
        for (i = 0; i < matrixSize; i++)

```

```

        for (k = 0; k < matrixSize; k++)
        for (j = 0; j < matrixSize; j++) {
            matrixRes[i][j] += (double)matrix1[i][k] * matrix2[k][j];
        }
    }
    else {
        printf("Error in MatrixMul(), wrong mulType_i");
    }
    ///вычисление времени:
    stop = clock();
    time_i += stop - start; ///время в тактах
    time_d = (double)time_i / CLOCKS_PER_SEC; ///время в секундах
    ///проверка:
    if (bCheck) {
        printf("\nafter multiplication: \n");
        printf("time_d=%f \n", time_d);
        PrintMatrix(matrix1, min((long long)3, matrixSize));
        printf("\n");
        PrintMatrix(matrix2, min((long long)3, matrixSize));
        printf("\n");
        PrintMatrix(matrixRes, min((long long)3, matrixSize));
        printf("\n");
    }
    ///освобождение памяти:
    for (i = 0; i < matrixSize; i++) {
        delete(matrix1[i]);
        delete(matrix2[i]);
        delete(matrixRes[i]);
    }
    delete(matrix1);
    delete(matrix2);
    delete(matrixRes);
    return 0;
}

int MatrixBlockMul(long long blockSize, long long matrixSize,
                   bool bCheck, double &time_d) {
    long long i, j, k;
    ///выделение памяти под матрицы:
    double *matrix1 = new double[matrixSize*matrixSize];
    double *matrix2 = new double[matrixSize*matrixSize];
    double *matrixRes = new double[matrixSize*matrixSize];
    ///заполнение матриц случайными числами:
    for (i = 0; i < matrixSize*matrixSize; i++) {
        matrix1[i] = rand() / 123456 + (double)rand() / RAND_MAX;
        matrix2[i] = rand() / 123456 + (double)rand() / RAND_MAX;
        matrixRes[i] = 0;
    }
    ///для вычисления времени:
    clock_t start, stop;
    long long time_i = 0;
    start = clock();
    ///умножение матриц по строке и столбцу:
    double *m1, *m2, *mRes; ///копии матриц
    long long i0, j0, k0;
    //long long blockSize = GetCacheAlignment();
    //printf("blockSize = %lld \n", blockSize);
    for (i = 0; i < matrixSize; i += blockSize)

```



```

for (j = 0; j < matrixSize; j += blockSize)
for (k = 0; k < matrixSize; k += blockSize) {
    for (i0 = 0, mRes = (matrixRes + i * matrixSize + j),
        m1 = (matrix1 + i * matrixSize + k); i0 < blockSize;
        ++i0, mRes += matrixSize, m1 += matrixSize)
    {
        for (k0 = 0, m2 = (matrix2 + k * matrixSize + j);
            k0 < blockSize; ++k0, m2 += matrixSize)
        {
            for (j0 = 0; j0 < blockSize; ++j0)
                mRes[j0] += m1[k0] * m2[j0];
        }
    }
}
}

///вычисление времени:
stop = clock();
time_i += stop - start; ///время в тактах
time_d = (double)time_i / CLOCKS_PER_SEC; ///время в секундах
///освобождение памяти:
delete(matrix1);
delete(matrix2);
delete(matrixRes);
return 0;
}

int WriteToCSV(char* mulType, long long launchCnt, long long matrixSize,
               long long blockSize,
               double avgTime, double absError, double relError)
{
    ///открытие output.csv:
    FILE *fout;
    if ((fout = fopen("../data/output.csv", "a")) == NULL) {
        printf("Error in Write_to_csv(): can't open output.csv \n");
        return 1;
    }

    fprintf(fout, "%s;", mulType);
    fprintf(fout, "%lld;", launchCnt);
    fprintf(fout, "%lld;", matrixSize);
    fprintf(fout, "%lld;", blockSize);
    fprintf(fout, "clock();"); ///Timer
    fprintf(fout, "%e;", avgTime);
    fprintf(fout, "%e;", absError);
    fprintf(fout, "%e%%;", relError);
    fprintf(fout, "\n");

    return 0;
}

int TestsHandler(char* mulType, long long launchCnt,
                 long long matrixSize, bool bCheck,
                 long long blockSize)
{
    ///начальные переменные для измерений:
    double summand1 = 0, summand2 = 0;
    double time_d[launchCnt];
    ///измерения (тестирование):

```

```

for (long long i = 0; i < launchCnt; i++) {
    if (strcmp("usual", mulType) == 0) {
        MatrixMul(1, matrixSize, bCheck, time_d[i]);
    }
    else if (strcmp("row_by_row", mulType) == 0) {
        MatrixMul(2, matrixSize, bCheck, time_d[i]);
    }
    else if (strcmp("block", mulType) == 0) {
        MatrixBlockMul(blockSize, matrixSize, bCheck, time_d[i]);
    }
    summand1 += time_d[i] * time_d[i];
    summand2 += time_d[i];
}
///заключительные переменные для измерений записи:
summand1 /= launchCnt;
summand2 /= launchCnt;
double avgTime = summand2; ///сумма времени всех тестов / n == среднее время
summand2 *= summand2;
double dispersion = summand1 - summand2; ///дисперсия (точность измерения
времени)
double absError = sqrt(dispersion); ///среднее квадратическое отклонение
(погрешность)
double relError = dispersion / avgTime * 100; ///относительная погрешность в
%

    WriteToCSV(mulType, launchCnt, matrixSize, blockSize, avgTime, absError,
relError);
    return 0;
}

```

Файл scripts/s2.sh

```

#!/bin/bash

#очистить output.csv и записать туда заданную строку
echo "multiplication type;launch count;matrix size;block size;timer;average
time;absError;relError;" > ../data/output.csv

cd ../source
make

for (( i=400; i<=2000; i+=400 ))
do
../source/main.exe --matrix-size $i --multiplication-type usual --launch-count
3
done

```

Файл scripts/d2.gpi

```

#!/usr/bin/gnuplot
#!/usr/bin/gnuplot -persist

#изображение, где будет диаграмма
set terminal png font "Verdana,12" size 1200, 800
set output "../data/diagram_t2.png"

#символ-раделитель в output.csv

```

```

set datafile separator ';'

#подпись осей
set xlabel "matrix size"
set ylabel "time in seconds"

#считать информацию, every 5 - считывать значение только из каждой 5 строки
plot "../data/output.csv" using 3:6 with linespoints lw 3 pt 5 title "usual mul"
#linespoints - линии с точками, lw - толщина линии, pt - размер точек

```

Файл scripts/s6.sh

```

#!/bin/bash

#очистить output.csv и записать туда заданную строку
echo "multiplication type;launch count;matrix size;block size;timer;average
time;absError;relError;" > ../data/output.csv

cd ../source
make

for (( i=256; i<=1024; i+=256 ))
do
../source/main.exe --matrix-size $i --multiplication-type usual --launch-count
3
../source/main.exe --matrix-size $i --multiplication-type row_by_row --launch-
count 3
../source/main.exe --matrix-size $i --multiplication-type block --launch-count
3
done

```

Файл scripts/d6.gpi

```

#!/usr/bin/gnuplot
#!/usr/bin/gnuplot -persist

#изображение, где будет диаграмма
set terminal png font "Verdana,12" size 1200, 800
set output "../data/diagram_t6.png"

#символ-разделитель в output.csv
set datafile separator ';'

#подпись осей
set xlabel "matrix size"
set ylabel "time in seconds"

#считать информацию, every 5 - считывать значение только из каждой 5 строки
plot "../data/output.csv" using 3:6 every 3::1 with linespoints lw 3 pt 5 title
"usual mul", \
    "../data/output.csv" using 3:6 every 3::2 with linespoints lw 3 pt 5 title
"row by row mul", \
    "../data/output.csv" using 3:6 every 3::3 with linespoints lw 3 pt 5 title
"block mul",

#linespoints - линии с точками, lw - толщина линии, pt - размер точек
#every 2::1 - брать значение каждые 2 строки, начиная с 1-й

```

```
#every 2::2 - брать значение каждые 2 строки, начиная с 2-й
#?? отсчет строк с 0, а столбцов с 1??
#сайт http://phaselockfel.blogspot.com/2007/08/gnuplot-to-plot-partial-data.html
```

Файл scripts/star1.sh

```
#!/bin/bash

#очистить output.csv и записать туда заданную строку
echo "multiplication type;launch count;matrix size;block size;timer;average
time;absError;relError;" > ../data/output.csv

cd ../source
make

for (( i=4; i<=256; i*=2 ))
do
../source/main.exe -s 512 -t block -l 3 --block-size $i
done
```

Файл scripts/star1.gpi

```
#!/usr/bin/gnuplot
#!/usr/bin/gnuplot -persist

#изображение, где будет диаграмма
set terminal png font "Verdana,12" size 1200, 800
set output "../data/diagram_star1.png"

#символ-разделитель в output.csv
set datafile separator ';'

#область значений для осей
set xrange [0:8]

#подписи по оси абсцисс
set xtics ("4" 1, "8" 2, "16" 3, "32" 4, "64" 5, "128" 6, "256" 7)

#подпись осей
set xlabel "block size (const matrix-size = 512)"
set ylabel "time in seconds"

#установить стиль гистограмм
set style data histograms

#установить ширину столбцов 0.8 от максимальной
set boxwidth 0.8 absolute
set style fill solid 1

#считать информацию, every 5 - считывать значение только из каждой 5 строки
plot "../data/output.csv" using 6 with boxes title "block mul"
```