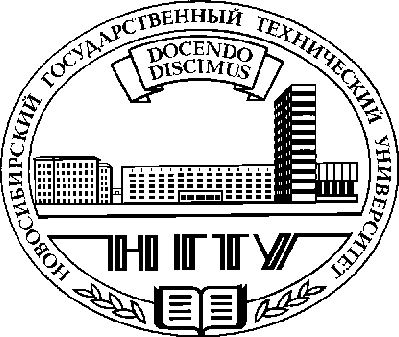
МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра вычислительной техники



**ОТЧЁТ**

**по лабораторной работе №3**

**по дисциплине: «***Вычислительные системы***»**

Выполнили: Преподаватель:

Студенты гр. АММ2-21, АВТФ Перышкова Е.Н.

Антонов С.С.

Арнольд Э.В.

Новосибирск

**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 числомстрок/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. Обеспечитьвозможность задавать блока, в качестве аргумента функции.

6. \*\* Реализовать дополнительную функцию DGEMM\_opt\_3, в которой выполняетсяоптимизация доступа к памяти,за счет векторизации кода.

7. Оценить ускорение умножения для матриц фиксированного размера, например,1000х1000, 2000х2000, 5000х5000, 10000х10000. \* 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 отчет отражающий суть, этапы и результаты проделанной работы.

**Код:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/time.h>

#include <string.h>

#include <math.h>

//вычисление работы оперативной памяти

int RAMw(long int number\_of\_elements)

{

    double write\_time = 0;

    struct timespec ts2, ts1;

    int buffer;

    srand(time(0));

    int \*A;

    A = malloc( sizeof(\*A) \* number\_of\_elements);

    for (int i = 0; i < number\_of\_elements; i++)

    {

        buffer = rand() % 1000;

        clock\_gettime(CLOCK\_REALTIME, &ts1);

        A[i] = buffer;

        clock\_gettime(CLOCK\_REALTIME, &ts2);

        write\_time += 1000000000 \* (ts2.tv\_sec - ts1.tv\_sec) + (ts2.tv\_nsec - ts1.tv\_nsec);

    }

    return write\_time;

}

int RAMr(long int number\_of\_elements)

{

    double read\_time = 0;

    struct timespec ts2, ts1;

    srand(time(0));

    int \*A;

    A = malloc( sizeof(\*A) \* number\_of\_elements);

    for (int i = 0; i < number\_of\_elements; i++)

    {

        clock\_gettime(CLOCK\_REALTIME, &ts1);

        A[i];

        clock\_gettime(CLOCK\_REALTIME, &ts2);

        read\_time += 1000000000 \* (ts2.tv\_sec - ts1.tv\_sec) + (ts2.tv\_nsec - ts1.tv\_nsec);

    }

    return read\_time;

}

//работа памятью

int memoryR(long int number\_of\_elements, char \*path)

{

    double read\_time = 0;

    srand(time(0));

    int ch;

    struct timespec ts2, ts1;

    FILE \*fp;

    fp = fopen(path, "r");

    if ( ! fp )

        return 1;

    clock\_gettime(CLOCK\_REALTIME, &ts1);

    while ( ( ch = fgetc(fp) ) != EOF );

    clock\_gettime(CLOCK\_REALTIME, &ts2);

    read\_time += 1000000000 \* (ts2.tv\_sec - ts1.tv\_sec) + (ts2.tv\_nsec - ts1.tv\_nsec);

    fclose(fp);

    return read\_time;

}

int memoryW(long int number\_of\_elements, char \*path)

{

    double write\_time = 0;

    int \*buffer;

    buffer = malloc( sizeof(\*buffer) \* number\_of\_elements);

    struct timespec ts2, ts1;

    FILE \*fp;

    for (int i = 0; i < number\_of\_elements; i++)

        buffer[i] = rand() % 1000;

    fp = fopen(path, "w");

    if ( ! fp )

        return 1;

    for (int i = 0; i < number\_of\_elements; i++)

    {

        clock\_gettime(CLOCK\_REALTIME, &ts1);

        fprintf(fp, "%d", buffer[i]);

        clock\_gettime(CLOCK\_REALTIME, &ts2);

        write\_time += 1000000000 \* (ts2.tv\_sec - ts1.tv\_sec) + (ts2.tv\_nsec - ts1.tv\_nsec);

    }

    fclose(fp);

    return write\_time;

}

// вычисления данных

void Math(int n, char \*operation, long int number\_of\_elements)

{

    double averageW = 0,  absErrorW = 0, relErrorW = 0, writeBandwidth = 0;

    double read\_time = 0, write\_time = 0;

    double dispersion = 0;

    double \*testArR;

    double \*testArW;

    testArR = malloc( sizeof(\*testArR) \* n);

    testArW = malloc( sizeof(\*testArW) \* n);

    double averageR = 0, absErrorR = 0, relErrorR = 0;

    double readBandwidth = 0;

    if(strcmp (operation, "RAM") == 0)

    {

        write\_time = RAMw(number\_of\_elements);

        read\_time = RAMr(number\_of\_elements);

        for(int i = 0; i < n; i++)

        {

            averageW += RAMw(number\_of\_elements);

            testArW[i] = RAMw(number\_of\_elements);

            averageR += RAMr(number\_of\_elements);

            testArR[i] = RAMr(number\_of\_elements);

        }

        averageR /= n;

        averageW /= n;

        for(int i = 0; i < n; i++)

            dispersion += pow(testArR[i] - averageR, 2) ;

        absErrorR = sqrt(dispersion / n);

        relErrorR = (absErrorR/ averageR) \* 100;

        readBandwidth = (number\_of\_elements / averageR) \* pow(10, 6);

        dispersion = 0;

        for(int i = 0; i < n; i++)

            dispersion += pow(testArW[i] - averageW, 2);

        absErrorW = sqrt(dispersion / n);

        relErrorW = (absErrorW/ averageW) \* 100;

        writeBandwidth = (number\_of\_elements / averageW) \* pow(10, 6);

        dispersion = 0;

    }

    else if(strcmp (operation, "SSD") == 0)

    {

        write\_time = memoryW(number\_of\_elements, "txt.txt");

        read\_time = memoryR(number\_of\_elements, "txt.txt");

        remove ("txt.txt");

        for(int i = 0; i < n; i++)

        {

            testArW[i] = memoryW(number\_of\_elements, "txt.txt");

            averageW += memoryW(number\_of\_elements, "txt.txt");

            averageR += memoryR(number\_of\_elements, "txt.txt");

            testArR[i] = memoryR(number\_of\_elements, "txt.txt");

            remove ("txt.txt");

        }

        averageR /= n;

        averageW /= n;

        for(int i = 0; i < n; i++)

            dispersion += pow(testArR[i] - averageR, 2) ;

        absErrorR = sqrt(dispersion / n);

        relErrorR = (absErrorR/ averageR) \* 100;

        readBandwidth = (number\_of\_elements / averageR) \* pow(10, 6);

        dispersion = 0;

        for(int i = 0; i < n; i++)

            dispersion += pow(testArW[i] - averageW, 2) ;

        absErrorW = sqrt(dispersion / n);

        relErrorW = (absErrorW/ averageW) \* 100;

        writeBandwidth = (number\_of\_elements / averageW) \* pow(10, 6);

        dispersion = 0;

    }

    else if(strcmp (operation, "flash") == 0)

    {

        write\_time = memoryW(number\_of\_elements, "/run/media/fuki/'ADATA UFD'/txt.txt");

        read\_time = memoryR(number\_of\_elements, "/run/media/fuki/'ADATA UFD'/txt.txt");

        remove ("/run/media/fuki/'ADATA UFD'/txt.txt");

        for(int i = 0; i < n; i++)

        {

            testArW[i]  = memoryW(number\_of\_elements, "/run/media/fuki/'ADATA UFD'/txt.txt");

            averageW += memoryW(number\_of\_elements, "/run/media/fuki/'ADATA UFD'/txt.txt");

            averageR += memoryR(number\_of\_elements, "/run/media/fuki/'ADATA UFD'/txt.txt");

            testArR[i] =  memoryR(number\_of\_elements, "/run/media/fuki/'ADATA UFD'/txt.txt");

            remove ("/run/media/fuki/'ADATA UFD'/txt.txt");

        }

        averageR /= n;

        averageW /= n;

        for(int i = 0; i < n; i++)

            dispersion += pow(testArR[i] - averageR, 2);

        absErrorR = sqrt(dispersion / n);

        relErrorR = (absErrorR/ averageR) \* 100;

        readBandwidth = (number\_of\_elements / averageR) \* pow(10, 6);

        dispersion = 0;

        for(int i = 0; i < n; i++)

            dispersion += pow(testArW[i] - averageW, 2);

        absErrorW = sqrt(dispersion / n);

        relErrorW = (absErrorW/ averageW) \* 100;

        writeBandwidth = (number\_of\_elements / averageW) \* pow(10, 6);

        dispersion = 0;

    }

    FILE  \*file;

    free(testArR);

    free(testArW);

    file = fopen("itog1.cvs", "a");

    fprintf(file, "\nMemoryType: %s , BlockSize: %ld, ElementType: int, BufferSize: , Timer: clock\_gettime(),  ", operation, number\_of\_elements);

    fprintf(file, "WriteTime: %.3f , AverageWriteTime: %.3f , WriteBandwidth: %.3f , AbsError(write): %.3f , RelError(write): %.3f  ", write\_time, averageW, writeBandwidth, absErrorW, relErrorW);

    fprintf(file, "ReadTime: %.3f , AverageReadTime: %.3f , ReadBandwidth: %.3f , AbsError(read): %.3f , RelError(read): %.3f ", read\_time, averageR, readBandwidth, absErrorR, relErrorR);

    fclose(file);

}

/\*

    "-m"

    "-b"

    "-l"

    "RAM"

    "SSD"

    "flash"

    "Kb"

    "Mb"

\*/

int main(int argc, char \*\*argv)

{

    long int number\_of\_elements;

    if (argc >= 4)

    {

        if (strcmp (argv[1], "m") == 0)

        {

            if(strcmp (argv[2], "RAM") == 0)

            {

                if (strcmp (argv[4], "Kb") == 0)

                    number\_of\_elements = atoi(argv[3]) \* 1024;

                else if (strcmp (argv[4], "Mb") == 0)

                    number\_of\_elements = atoi(argv[3]) \* 1024 \* 1024;

                else

                    number\_of\_elements = atoi(argv[3]);

                int n;

                if (strcmp (argv[5], "l") == 0)

                    n = atoi(argv[6]);

                else

                    n = atoi(argv[5]);

                printf("%ld\n", number\_of\_elements);

                printf("%d\n", n);

                Math(n, "SSD", number\_of\_elements);

            }

            else if(strcmp (argv[2], "SDD") == 0)

            {

                if (strcmp (argv[4], "Kb") == 0)

                    number\_of\_elements = atoi(argv[3]) \* 1024;

                else if (strcmp (argv[4], "Mb") == 0)

                    number\_of\_elements = atoi(argv[3]) \* 1024 \* 1024;

                else

                    number\_of\_elements = atoi(argv[3]);

                int n;

                if (strcmp (argv[5], "l") == 0)

                    n = atoi(argv[6]);

                else

                    n = atoi(argv[5]);

                printf("%ld\n", number\_of\_elements);

                printf("%d\n", n);

                Math(n, "SSD", number\_of\_elements);

            }

            else if(strcmp (argv[2], "flash") == 0)

            {

                if (strcmp (argv[4], "Kb") == 0)

                    number\_of\_elements = atoi(argv[3]) \* 1024;

                else if (strcmp (argv[4], "Mb") == 0)

                    number\_of\_elements = atoi(argv[3]) \* 1024 \* 1024;

                else

                    number\_of\_elements = atoi(argv[3]);

                int n;

                if (strcmp (argv[5], "l") == 0)

                    n = atoi(argv[6]);

                else

                    n = atoi(argv[5]);

                Math(n, "flash", number\_of\_elements);

            }

            else

                printf("Неправильный ввод.\n" );

        }

    }

    return 0;

}

**Результаты работы :**



