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

КАФЕДРА ВС

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

«Многопоточное программирование»

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

1. Для программы умножения двух квадратных матриц DGEMM BLAS разработанной в задании 4 на языке C/C++ реализовать многопоточные вычисления. В потоках необходимо реализовать инициализацию массивов случайными числами типа double и равномерно распределить вычислительную нагрузку. Обеспечить возможность задавать размерность матриц и количество потоков при запуске программы. Многопоточность реализовать несколькими способами.

    1) С использованием библиотеки стандарта POSIX Threads.

    2) С использованием библиотеки стандарта OpenMP.

    3) * С использованием библиотеки Intel TBB.

    4) ** С использованием библиотеки стандарта MPI. Все матрицы помещаются в общей памяти одного вычислителя.

    5) *** С использованием технологий многопоточности для графических сопроцессоров (GPU) - CUDA/OpenCL/OpenGL/OpenACC.

2. Для всех способов организации многопоточности построить график зависимости коэффициента ускорения многопоточной программы от числа потоков для заданной размерностиматрицы, например, 5000, 10000 и 20000 элементов.

3. Определить оптимальное число потоков для вашего оборудования.

4. Подготовить отчет отражающий суть, этапы и результаты проделанной работы.

Выполнение работы

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.

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;
- MatrixMul() - в этой функции происходит умножение матриц в зависимости от переданных параметров;
- MatrixMulForThread() - инициализирующая функция для POSIX threads;
- WriteToCSV() для записи результатов тестирования в файл output.csv;
- TestsHandler() для запуска нужного умножения launch-count раз и вычисления времени и погрешностей.

Тестирование осуществляется с помощью Bash- и gnuplot-скриптов, которые запускают программу с нужными параметрами. Полученные результаты записываются в csv-файл. На данных этого файла gnuplot-скрипт создает изображение.

Результат работы

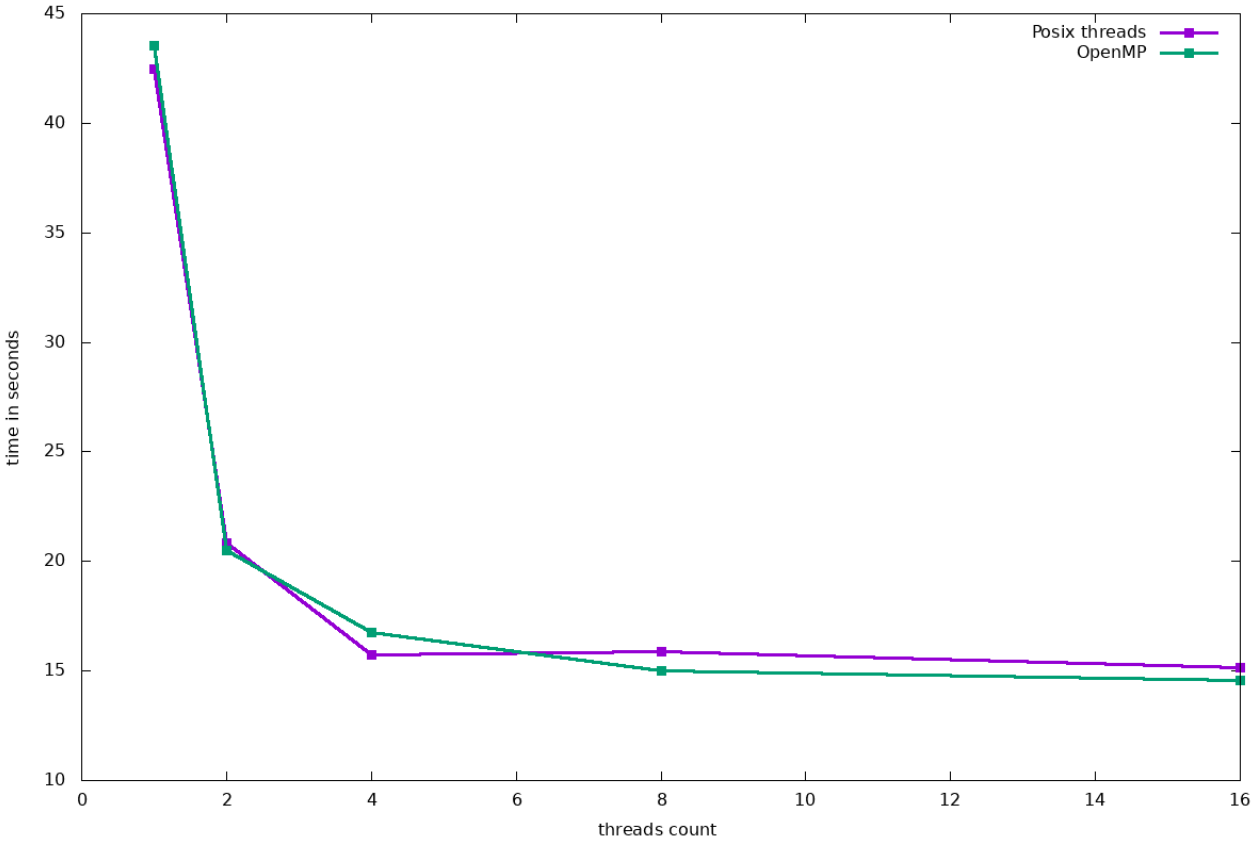

Рисунок 1. График зависимости количества потоков и времени при размере матрицы 1024 на 1024 элементов.

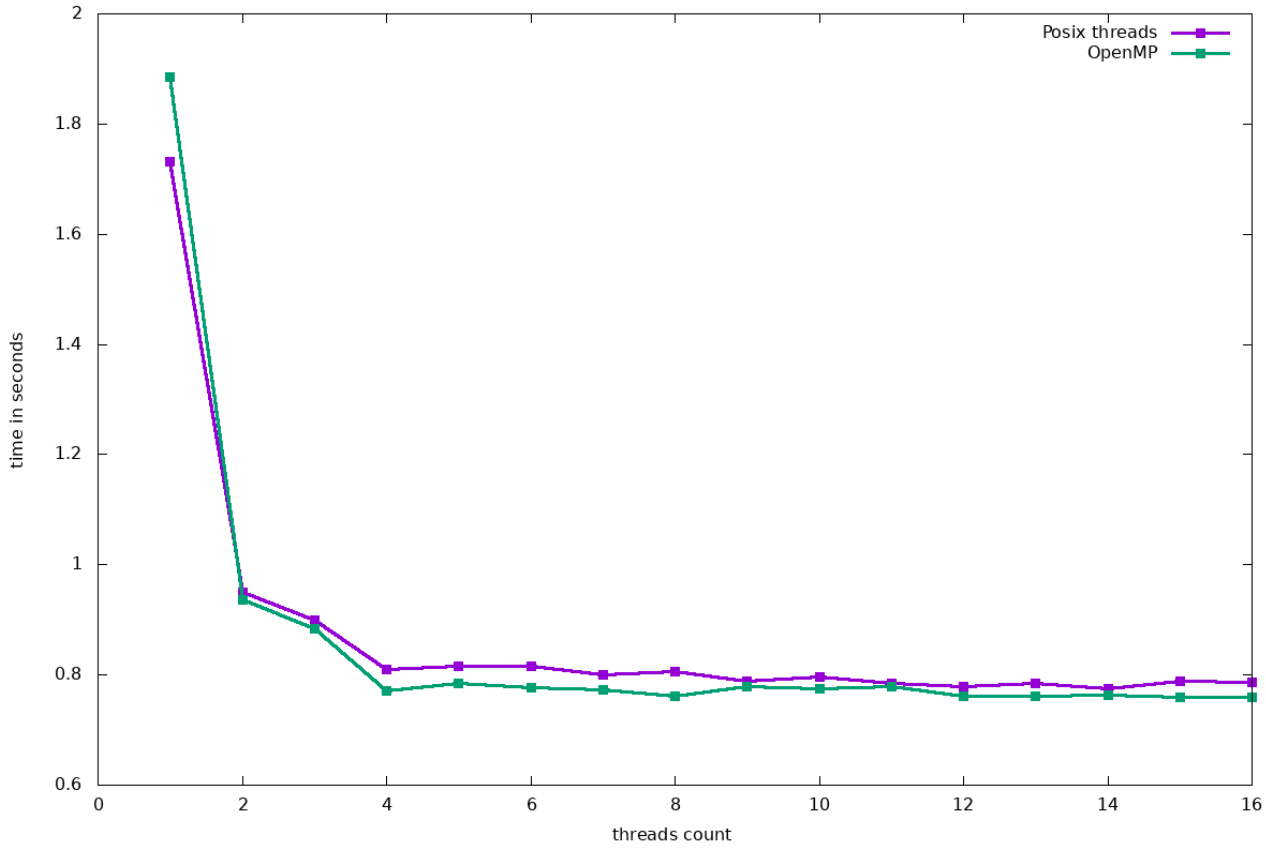

Рисунок 2. График зависимости количества потоков и времени при размере матрицы 512 на 512 элементов.

Примечание: Как видно на рисунках 1 и 2 оптимальное число потоков — 4, т. к. при 4-ех потоках достигается минимальное время. Для большего количества потоков время почти не меняется.

multiplication type	launch count	matrix size	block size	threadsCnt	timer	average time	absError	relError
POSIX_Threads	5	512	64	1	clock_gettime()	1.732377e+00	2.421128e-01	3.383710e+00%
OpenMP	5	512	64	1	clock_gettime()	1.885439e+00	4.754206e-02	1.198791e-01%
POSIX_Threads	5	512	64	2	clock_gettime()	9.499061e-01	1.680249e-02	2.972123e-02%
OpenMP	5	512	64	2	clock_gettime()	9.369529e-01	1.763556e-02	3.319407e-02%
POSIX_Threads	5	512	64	3	clock_gettime()	8.987253e-01	1.554389e-02	2.688391e-02%
OpenMP	5	512	64	3	clock_gettime()	8.846584e-01	1.158612e-02	1.517401e-02%
POSIX_Threads	5	512	64	4	clock_gettime()	8.103163e-01	2.331024e-02	6.705620e-02%
OpenMP	5	512	64	4	clock_gettime()	7.715133e-01	3.296838e-03	1.408808e-03%
POSIX_Threads	5	512	64	5	clock_gettime()	8.158584e-01	1.238376e-02	1.879707e-02%
OpenMP	5	512	64	5	clock_gettime()	7.844456e-01	1.949785e-02	4.846302e-02%
POSIX_Threads	5	512	64	6	clock_gettime()	8.148935e-01	1.633547e-02	3.274633e-02%
OpenMP	5	512	64	6	clock_gettime()	7.775438e-01	7.583857e-03	7.396997e-03%
POSIX_Threads	5	512	64	7	clock_gettime()	8.001095e-01	8.169384e-03	8.341212e-03%
OpenMP	5	512	64	7	clock_gettime()	7.721312e-01	1.951168e-02	4.930581e-02%
POSIX_Threads	5	512	64	8	clock_gettime()	8.053909e-01	4.645021e-03	2.678975e-03%
OpenMP	5	512	64	8	clock_gettime()	7.614746e-01	8.375448e-03	9.212144e-03%
POSIX_Threads	5	512	64	9	clock_gettime()	7.890615e-01	1.475638e-02	2.759618e-02%
OpenMP	5	512	64	9	clock_gettime()	7.785748e-01	4.196386e-02	2.261781e-01%

Рисунок 3. Первые строки csv-файла.

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

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

Файл source/foo.h

```
#ifndef FOO
#define FOO

#include <math.h> //для вычисления погрешностей
#include <stdio.h> //ввод/вывод
#include <stdlib.h> //рандом
#include <time.h> //измерение времены
#include <string.h> //Для сравнения строк
#include <pthread.h> //POSIX threads
#include <omp.h> //OpemMP

typedef struct a {
    double *matrix1;
    double *matrix2;
    double *matrixRes;
    int matrixSize;
    int from;
    int to;
} argsForThread;
```

```
int ProcessParameters(int argc, char *argv[],
                      int &matrixSize, char* mulType,
                      int &launchCnt, bool &bCheck,
                      int &blockSize, int &threadsCnt);
int TestsHandler(char* mulType, int launchCnt,
                 int matrixSize, bool bCheck,
                 int blockSize, int &threadsCnt);
long long GetCacheAlignment();

#endif
```

Файл source/foo.cpp

```
#include "foo.h"

int ProcessParameters(int argc, char *argv[],
                      int &matrixSize, char* mulType,
                      int &launchCnt, bool &bCheck,
                      int &blockSize, int &threadsCnt)
{
    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 = atoi(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 ||
                strcmp("POSIX_Threads", argv[i]) == 0 ||
                strcmp("OpenMP", 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 = atoi(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 = atoi(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;
            }
        }
        ///если число потоков:
        else if (strcmp("-tc", argv[i]) == 0 ||
                 strcmp("--threads-count", argv[i]) == 0)
        {
            i++;
            threadsCnt = atoi(argv[i]); ///приведение строки в long long int
            if (threadsCnt <= 0) {
                  printf("Error in arguments of main(): incorrect value for --
threads-count \n");
                return 1;
            }
        }
    }

    return 0;
}

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);
    int val = atoi(temp);
    if (val == 0) {
        printf("Error in GetCacheAlignment(): can't atoll \n");
        return -1;
    }

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

int PrintMatrix(double *matrix, int n) {
    int i, j;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            //printf("[%lld][%lld]=", i, j);
            printf("%.2f ", matrix[i * n + j]);
        }
        printf("\n");
    }
    return 0;
}

void* MatrixMulForThread(void* voidpArgs)
{
    argsForThread* pArgs = (argsForThread*) voidpArgs;

    for (int i = pArgs->from; i < pArgs->to; i++) {
        //printf("i=%d ", i);
        for (int j = 0; j < pArgs->matrixSize; j++)
        for (int k = 0; k < pArgs->matrixSize; k++) {
            pArgs->matrixRes[i * pArgs->matrixSize + j]  +=
                pArgs->matrix1[i * pArgs->matrixSize + k] *
                pArgs->matrix2[k * pArgs->matrixSize + j];
        }
    }

    return 0;
}

int MatrixMul(int mulType_i, int blockSize,
              int matrixSize, bool bCheck,
              double &time_d, int threadsCnt)
{
    int 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:
    if (bCheck) for (i = 0; i < matrixSize*matrixSize; i++) {
```

```
    matrix1[i] = i;
    matrix2[i] = i;
    matrixRes[i] = 0;
}
else 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;
}
///проверка:
if (bCheck) {
    printf("\nmatrix1: \n");
    PrintMatrix(matrix1, matrixSize);
    printf("\nmatrix2: \n");
    PrintMatrix(matrix2, matrixSize);
    printf("\n");
}
///для вычисления времени:
struct timespec mt1, mt2;
clock_gettime(CLOCK_REALTIME, &mt1);
///--- умножение матриц: ---///
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 * matrixSize + j]  +=
            matrix1[i * matrixSize + k] * matrix2[k * matrixSize + j];
    }
}
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 * matrixSize + j]  +=
            matrix1[i * matrixSize + k] * matrix2[k * matrixSize + j];
    }
}
if (mulType_i == 3) { ///блочное умножение матриц
    double *m1, *m2, *mRes; ///копии матриц
    int i0, j0, k0;
    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];
            }
        }
    }
}
if (mulType_i == 4) { ///POSIX Threads
    ///получить дефолтные значения атрибутов потоков:
```

```
        pthread_attr_t attr;
        pthread_attr_init(&attr);
        ///указатель на указатели структуры argsForThread:
              argsForThread**  pArgs  =  (argsForThread**)  malloc(threadsCnt  *
sizeof(argsForThread*));
        pthread_t thread_id[threadsCnt]; ///идентификаторы потоков
        int statuses[threadsCnt]; ///
        int statuses_sum = 0;
        ///определние "границ" для каждого потока:
        for (int i = 0; i < threadsCnt; i++) {
            pArgs[i] = (argsForThread*) malloc(sizeof(argsForThread));
            pArgs[i]->matrix1 = matrix1;
            pArgs[i]->matrix2 = matrix2;
            pArgs[i]->matrixRes = matrixRes;
            pArgs[i]->matrixSize = matrixSize;
            pArgs[i]->from = matrixSize / threadsCnt * i;
            pArgs[i]->to = matrixSize / threadsCnt * (i + 1);
            ///создание потока:
            pthread_create(&thread_id[i], &attr, MatrixMulForThread, pArgs[i]);
        }
        for (int i = 0; i < threadsCnt; i++) {
            ///подождать поток:
            statuses[i] = pthread_join(thread_id[i], NULL);
            statuses_sum += statuses[i];
        }
        ///проверка:
        if (statuses_sum != 0)
            printf("error, MatrixMulForThread() failed \n");
        ///очистка памяти:
        for (int i = 0; i < threadsCnt; i++) {
            free(pArgs[i]);
        }
        free(pArgs);
    }
    if (mulType_i == 5) { ///OpenMP
#pragma omp parallel for shared(matrix1, matrix2, matrixRes) \
private(j, k) num_threads(threadsCnt) schedule(static)
        for (i = 0; i < matrixSize; i++) {
            for (j = 0; j < matrixSize; j++) {
                //double tmp = 0;
                for (k = 0; k < matrixSize; k++) {
                    matrixRes[i * matrixSize + j] +=
                    matrix1[i * matrixSize + k] * matrix2[k * matrixSize + j];
                }
                //matrixRes[i * matrixSize + j] = tmp;
            }
        }
    }
    ///проверка:
    if (bCheck) {
        printf("\nmatrixRes: \n");
        PrintMatrix(matrixRes, matrixSize);
    }
    ///вычисление времени:
    clock_gettime(CLOCK_REALTIME, &mt2);
    time_d = (double)(mt2.tv_sec - mt1.tv_sec) +
             (double)(mt2.tv_nsec - mt1.tv_nsec) / 1e9; ///время в секундах
    ///освобождение памяти:
    delete(matrix1);
```

```
    delete(matrix2);
    delete(matrixRes);
    return 0;
}


int WriteToCSV(char* mulType, int launchCnt, int matrixSize,
               int blockSize, int threadsCnt,
               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, "%d;", launchCnt);
    fprintf(fout, "%d;", matrixSize);
    fprintf(fout, "%d;", blockSize);
    fprintf(fout, "%d;", threadsCnt);
    fprintf(fout, "clock_gettime();"); ///Timer
    fprintf(fout, "%e;", avgTime);
    fprintf(fout, "%e;", absError);
    fprintf(fout, "%e%%;", relError);
    fprintf(fout, "\n");

    return 0;
}

int TestsHandler(char* mulType, int launchCnt,
                 int matrixSize, bool bCheck,
                 int blockSize, int &threadsCnt)
{
    ///начальные переменные для измерений:
    double summand1 = 0, summand2 = 0;
    double time_d[launchCnt];
    ///измерения (тестирование):
    for (int i = 0; i < launchCnt; i++) {
        if (strcmp("usual", mulType) == 0) {
            MatrixMul(1, blockSize, matrixSize, bCheck, time_d[i], threadsCnt);
        }
        if (strcmp("row_by_row", mulType) == 0) {
            MatrixMul(2, blockSize, matrixSize, bCheck, time_d[i], threadsCnt);
        }
        if (strcmp("block", mulType) == 0) {
            MatrixMul(3, blockSize, matrixSize, bCheck, time_d[i], threadsCnt);
        }
        if (strcmp("POSIX_Threads", mulType) == 0) {
            MatrixMul(4, blockSize, matrixSize, bCheck, time_d[i], threadsCnt);
        }
        if (strcmp("OpenMP", mulType) == 0) {
            MatrixMul(5, blockSize, matrixSize, bCheck, time_d[i], threadsCnt);
        }
        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, threadsCnt, avgTime,
absError, relError);
    return 0;
}
```

Файл scripts/threads_s1024.sh

```
#!/bin/bash

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

cd ../source
make

for (( i=1; i<=16; i*=2 ))
do
./../source/main.exe --matrix-size 1024 --multiplication-type POSIX_Threads \
                     --threads-count $i --launch-count 3
./../source/main.exe --matrix-size 1024 --multiplication-type OpenMP \
                     --threads-count $i --launch-count 3
done
```

Файл scripts/threads_s512.sh

```
#!/bin/bash

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

cd ../source
make

for (( i=1; i<=16; i+=1 ))
do
./../source/main.exe --matrix-size 512 --multiplication-type POSIX_Threads \
                     --threads-count $i --launch-count 5
./../source/main.exe --matrix-size 512 --multiplication-type OpenMP \
                     --threads-count $i --launch-count 5
done
```

Файл scripts/threads_s.gpi

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

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

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

#подпись осей
set xlabel "threads count"
set ylabel "time in seconds"

#считать информацию
plot "../data/output.csv" using 5:7 every 2::1 with linespoints lw 3 pt 5 title
"Posix threads", \
     "../data/output.csv" using 5:7 every 2::2 with linespoints lw 3 pt 5 title
"OpenMP"

#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
```