Федеральное Государственное Бюджетное Образовательное Учреждение

«Сибирский Государственный Университет Телекоммуникаций и Информатики»

Лабораторная работа №4

по дисциплине

Автоматизация Вычислительных Систем

Выполнил: студент

группы ИП-814

Краснов Илья

Проверил: ст. преп. Кафедры ВС   
Ткачева Т. А.

Оглавление

[**Постановка задачи** 3](#_Toc58187077)

[**Реализация** 4](#_Toc58187078)

[**Листинг программы** 6](#_Toc58187079)

[**Ссылки** 9](#_Toc58187080)

# **Постановка задачи**

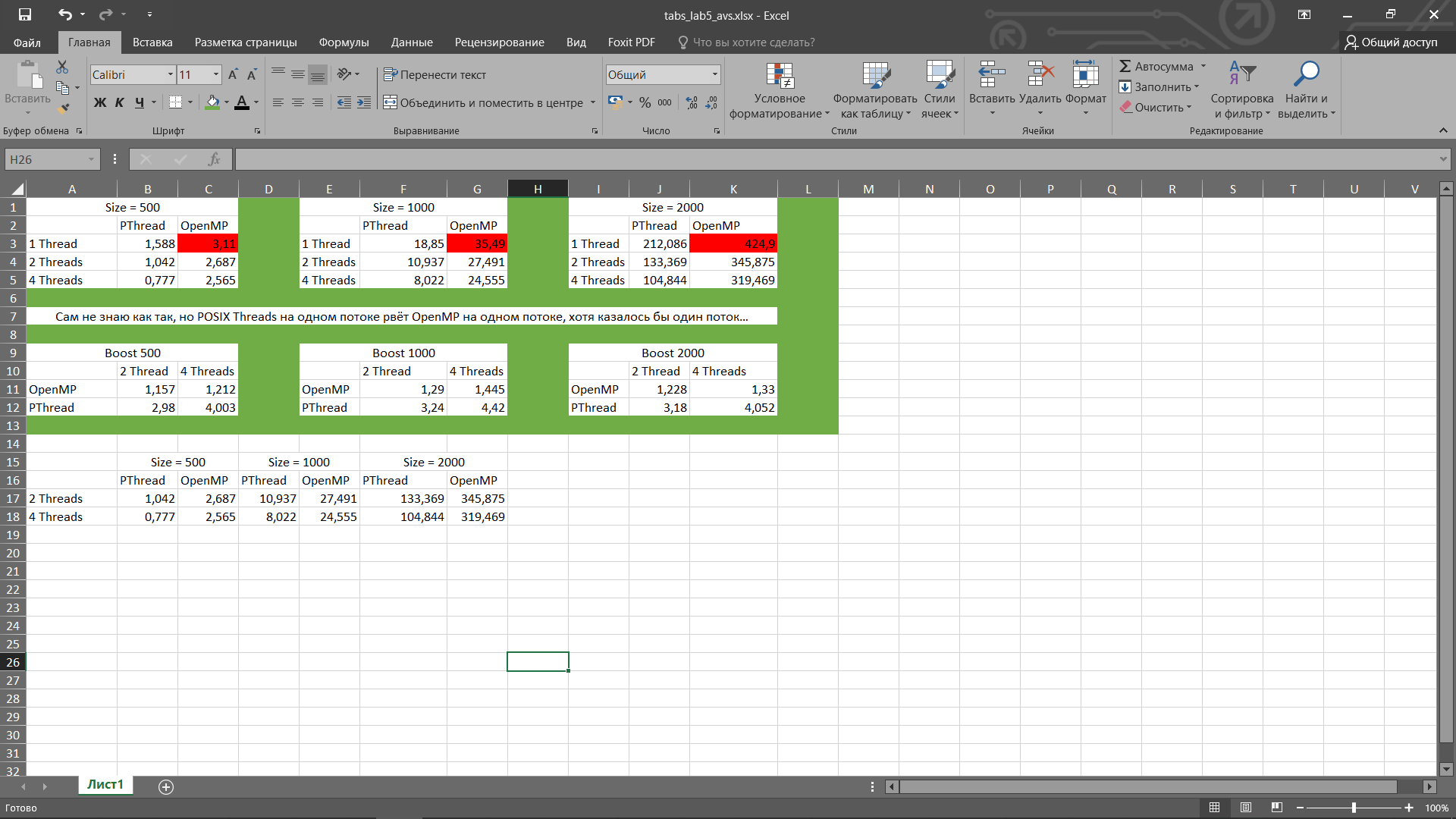
1. Для программы умножения двух квадратных матриц DGEMM BLAS разработанной в  
   задании 4 на языке С/С++ реализовать многопоточные вычисления. В потоках  
   необходимо реализовать инициализацию массивов случайными числами типа  
   double и равномерно распределить вычислительную нагрузку. Обеспечить  
   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.

1. Для всех способов организации многопоточности построить график зависимости  
   коэффициента ускорения многопоточной программы от числа потоков для заданной  
   размерности матрицы, например, 5000, 10000 и 20000 элементов.
2. Определить оптимальное число потоков для вашего оборудования.
3. Подготовить отчет отражающий суть, этапы и результаты проделанной работы.

# **Реализация**



(рис.1) Результаты работы программы

(рис.2) Зависимость коэффициента ускорения от числа потоков в матрице 500х500

(рис.3) Зависимость коэффициента ускорения от числа потоков в матрице 1000х1000

(рис.4) Зависимость коэффициента ускорения от числа потоков в матрице 2000х2000

# **Листинг программы**

#include <cmath>

#include <ctime>

#include <cstdlib>

#include <iomanip>

#include <iostream>

#include <pthread.h>

#include <omp.h>

*using* *namespace* std;

double timeStart;

double timeCounter;

double\*\* aMatrix;

double\*\* bMatrix;

double\*\* resMatrix;

void **BenchTimer**(string task)

{

*if* (task == "START"){

timeStart = clock();

} *else* *if*(task == "STOP"){

timeCounter = ((double)(clock() - timeStart)) / CLOCKS\_PER\_SEC;

}

}

void **FillMatrix**(int size)

{

srand(time(NULL));

*for* (int i = 0; i < size; i++){

*for* (int j = 0; j < size; j++){

aMatrix[i][j] = rand()%100;

bMatrix[i][j] = rand()%100;

}

}

}

void **PrintMatrix**(double\*\* matrix, int size)

{

*for* (int i = 0; i < size; i++){

*for* (int j = 0; j < size; j++){

cout << matrix[i][j] << " ";

}

cout << endl;

}

}

double **MultipicMatrix**(int size, int x\_pos, int y\_pos)

{

double res = 0;

int i = 0, j = 0;

*while* (i < size){

res += aMatrix[x\_pos][i] \* bMatrix[j][y\_pos];

i++;

j++;

}

*return* res;

}

void\* **DGEMM\_BLAS\_pt**(void\* args)

{

int start = atoi(((char\*\*)args)[0]);

int end = atoi(((char\*\*)args)[1]);

int size = atoi(((char\*\*)args)[2]);

*for* (int i = start; i < end; i++){

*for* (int j = 0; j < size; j++){

resMatrix[i][j] = MultipicMatrix(size, i, j);

}

}

pthread\_exit(0);

}

void **DGEMM\_BLAS\_omp**(int start, int end, int size) {

*for*(int i = start; i < end; i++) {

*for*(int j = 0; j < size; j++){

resMatrix[i][j] = MultipicMatrix(size, i, j);

}

}

}

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

{

*if* (argc < 2){

cout << "ERROR: Incorrrect number of elements"<< endl;

cout << "Elements: Matrix Size, Number of Threads"<<endl;

*//* *return* *0;*

}

int num\_of\_threads, matrix\_size;

matrix\_size = atoi(argv[1]);

num\_of\_threads = atoi(argv[2]);

*if* (num\_of\_threads > matrix\_size)

num\_of\_threads = matrix\_size;

aMatrix = *new* double\* [matrix\_size];

bMatrix = *new* double\* [matrix\_size];

resMatrix = *new* double\* [matrix\_size];

*for* (int i = 0; i < matrix\_size; i++){

aMatrix[i] = *new* double[matrix\_size];

bMatrix[i] = *new* double[matrix\_size];

resMatrix[i] = *new* double[matrix\_size];

}

char\*\* args = *new* char\* [3];

*for* (int i = 0; i < 3; i++)

args[i] = *new* char[10];

int step =(int)(matrix\_size / num\_of\_threads);

int start\_i = 0;

int stop\_i = step;

FillMatrix(matrix\_size);

pthread\_t\* threads = *new* pthread\_t[num\_of\_threads];

snprintf(args[2], 10, "%d", matrix\_size);

BenchTimer("START");

*for* (int i = 0; i < num\_of\_threads - 1; i++){

snprintf(args[0], 10, "%d", start\_i);

snprintf(args[1], 10, "%d", stop\_i);

pthread\_create(&threads[i], NULL, DGEMM\_BLAS\_pt, args);

start\_i = stop\_i;

stop\_i += step;

}

snprintf(args[0], 10, "%d", start\_i);

snprintf(args[1], 10, "%d", matrix\_size);

pthread\_create(&threads[num\_of\_threads - 1], NULL, DGEMM\_BLAS\_pt, args);

*for* (int i = 0; i < num\_of\_threads; i++){

pthread\_join(threads[i], NULL);

}

BenchTimer("STOP");

cout << "POSIX Threads Time: " << timeCounter;

BenchTimer("Start");

#pragma omp parallel

{

#pragma omp for

*for*(int i = 0; i < num\_of\_threads; i++) {

*if* (i == num\_of\_threads - 1) DGEMM\_BLAS\_omp(step \* (num\_of\_threads - 1), matrix\_size, matrix\_size);

*else* DGEMM\_BLAS\_omp(i \* step, (i + 1) \* step, matrix\_size);

}

}

BenchTimer("STOP");

cout << "OpenMP Time: " << timeCounter;

*delete* aMatrix;

*delete* bMatrix;

*delete* resMatrix;

*return* 0;

}

# **Ссылки**

Материалы для изучения POSIX Threads: [Unix2019b/Библиотека pthreads — iRunner Wiki (bsu.by)](https://acm.bsu.by/wiki/Unix2019b/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_pthreads)

Материалы для изучения OpenMP: [Что такое OpenMP? | PARALLEL.RU - Информационно-аналитический центр по параллельным вычислениям](https://parallel.ru/tech/tech_dev/openmp.html)

Репозиторий GitHub с исходным кодом и данными о текущем состоянием программы: [Study/AVS/Lab5 at master · Krasnoffsky/Study (github.com)](https://github.com/Krasnoffsky/Study/tree/master/AVS/Lab5)