Федеральное Государственное Бюджетное Образовательное Учреждение

«Сибирский Государственный Университет Телекоммуникаций и Информатики»

Лабораторная работа №5

по дисциплине

Автоматизация Вычислительных Систем

Выполнил: студент

группы ИП-814

Якунин А. В.

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

Оглавление

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

[**Реализация** 4](#_Toc58187078)

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

# **Постановка задачи**

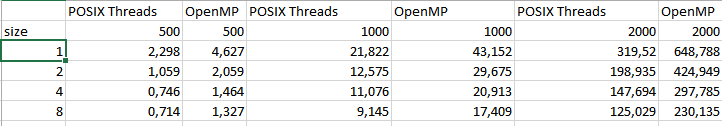
1. Для программы умножения двух квадратных матриц DGEMM BLAS разработанной в  
   задании 4 на языке С/С++ реализовать многопоточные вычисления. В потоках  
   необходимо реализовать инициализацию массивов случайными числами типа  
   double и равномерно распределить вычислительную нагрузку. Обеспечить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

1) С использованием библиотеки стандарта POSIX Threads.

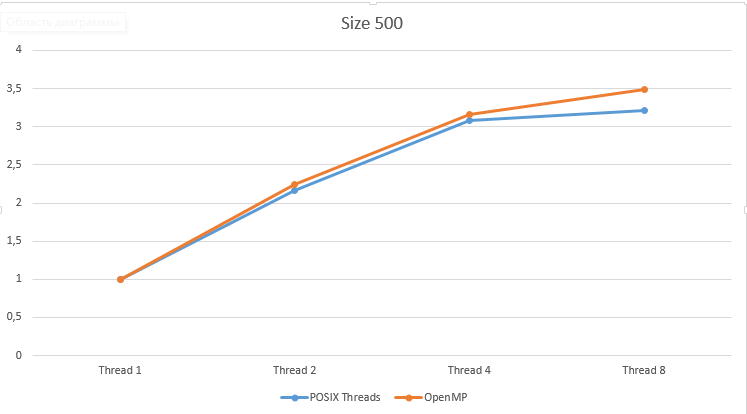
2) С использованием библиотеки стандарта OpenMP.

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.

# **Результаты тестирования**



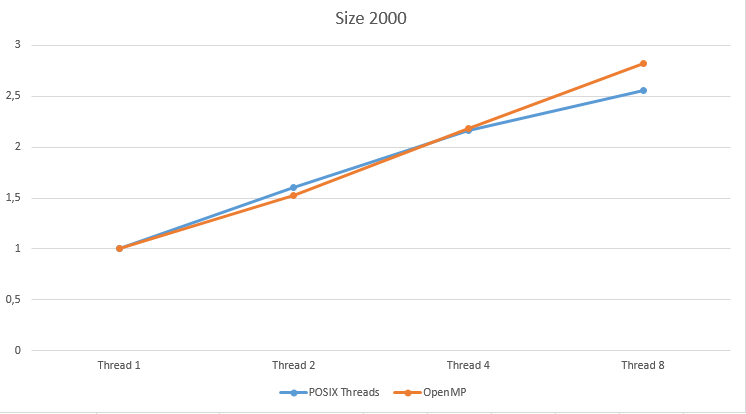
(рис.1) Результаты работы программы

****

(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

При тестировании использовался двухъядерный процессор Intel(R) Core(TM) i3-6006U CPU @ 2.00GHz. Оптимальное число потоков равно числу ядер. Для данного процессора оптимальное число потоков рано 2.

# **Листинг программы**

#include <cmath>

#include <ctime>

#include <cstdlib>

#include <iomanip>

#include <iostream>

#include <pthread.h>

#include <C:\Qt\Tools\mingw810\_64\lib\gcc\x86\_64-w64-mingw32\8.1.0\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 size)

{

int i = 0, j = 0;

#pragma omp parallel for private(i, j)

for (i = 0; i < size; i++)

for (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]);

omp\_set\_dynamic(0);

omp\_set\_num\_threads(num\_of\_threads);

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");

DGEMM\_BLAS\_omp(matrix\_size);

BenchTimer("STOP");

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

delete aMatrix;

delete bMatrix;

delete resMatrix;

return 0;

}