Министерство цифрового развития, связи и массовых коммуникаций

Российской Федерации Сибирский Государственный Университет

Телекоммуникаций и Информатики СибГУТИ

Кафедра Вычислительных систем

Лабораторная работа №5

По дисциплине “Архитектура вычислительных систем”

Выполнил:

Студент группы ИВ-921

Черемисин И.И.

Работу проверил:

Ассистент кафедры ВС

Петухова Я.В.

Новосибирск 2021

**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.
3. 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.
4. Определить оптимальное число потоков для вашего оборудования.
5. Подготовить отчет отражающий суть, этапы и результаты проделанной работы.

**Результаты работы**

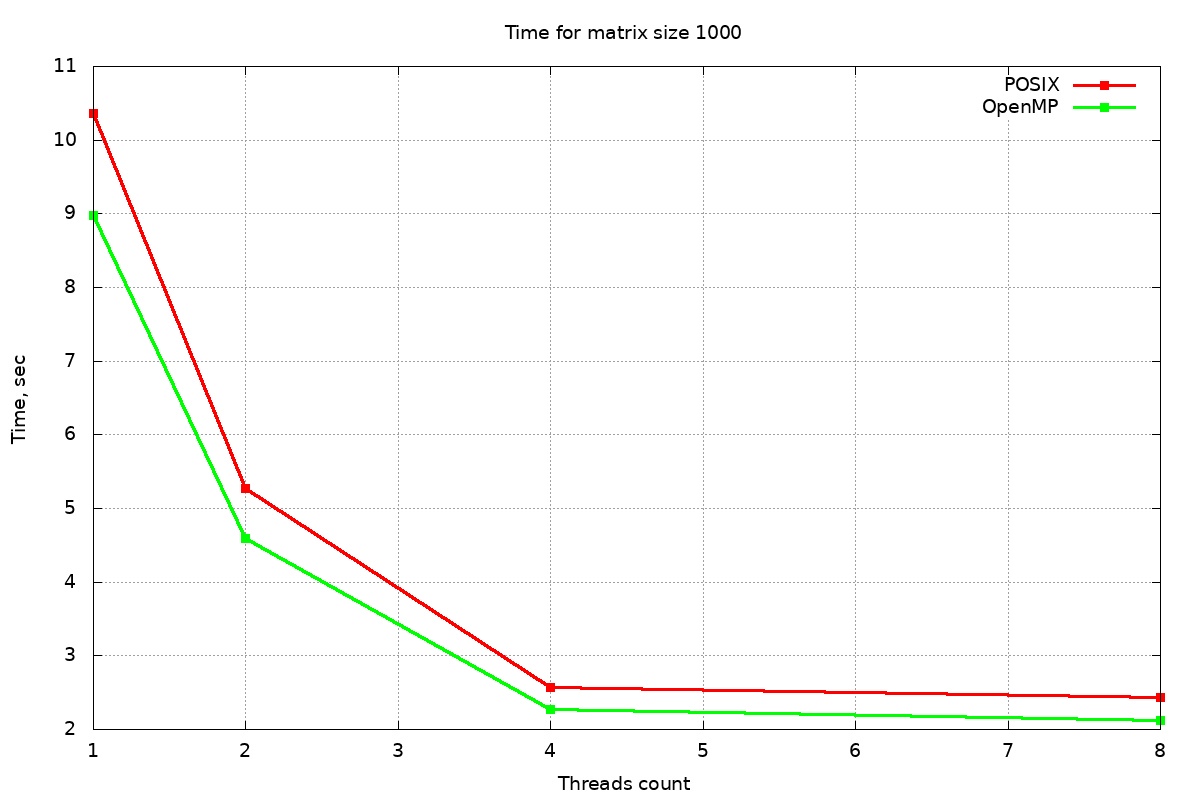


График 1. График зависимости количества потоков и времени при размере матрицы 1000 на 1000 элементов.

На Графике 1 оптимальное число потоков — 8, т. к. при этом кол-ве потоков достигается минимальное время выполнения программы.

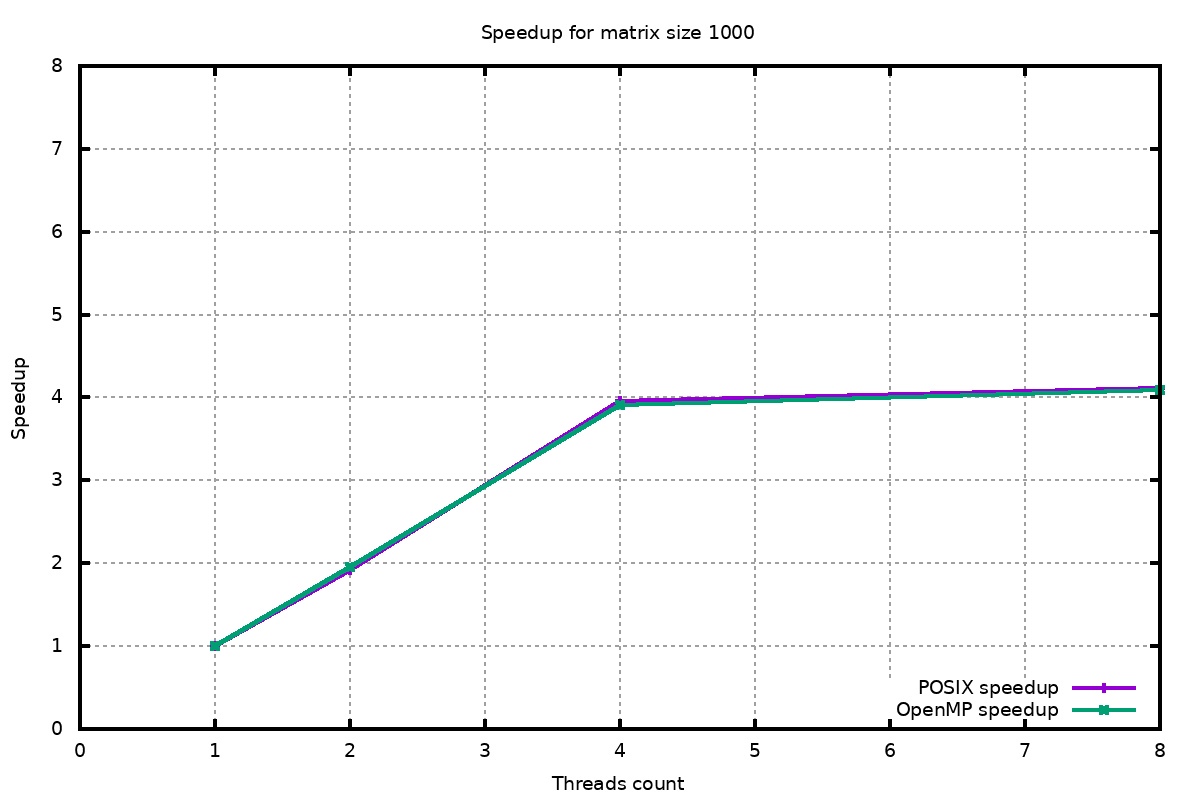


График 2. График зависимости ускорения многопоточной программы от числа потоков при размере матрицы 1000 на 1000 элементов.

На Графике 2 видно, что OpenMP и POSIX особо не отличаются.

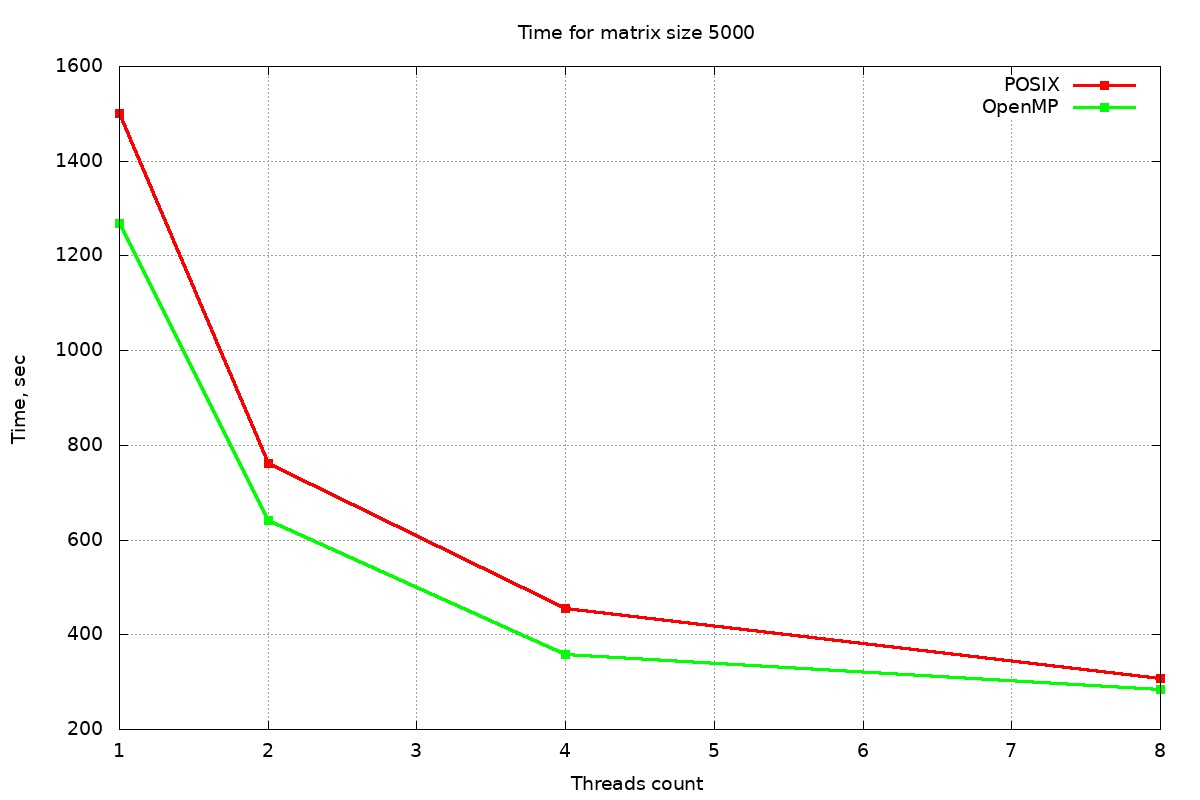


График 3. График зависимости количества потоков и времени при размере матрицы 5000 на 5000 элементов.

На Графике 3 оптимальное число потоков — 8, т. к. при этом кол-ве потоков достигается минимальное время выполнения программы.

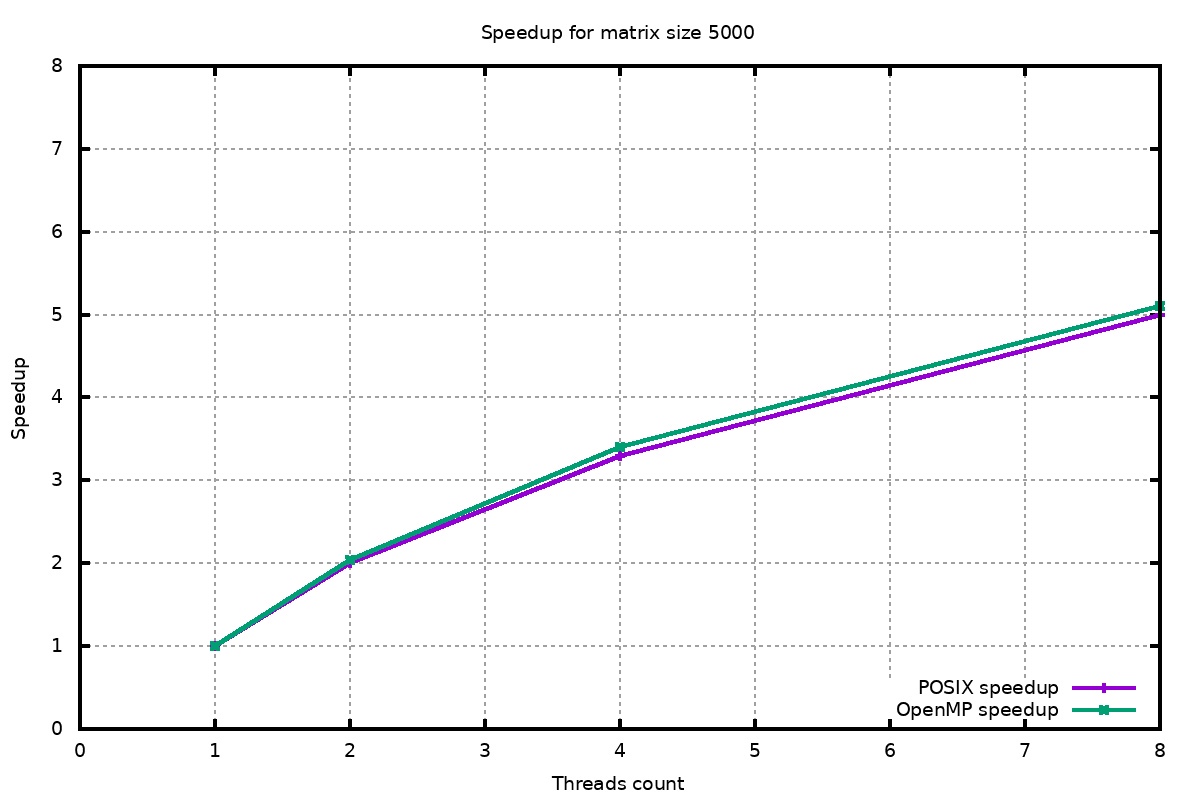


График 4. График зависимости ускорения многопоточной программы от числа потоков при размере матрицы 5000 на 5000 элементов.

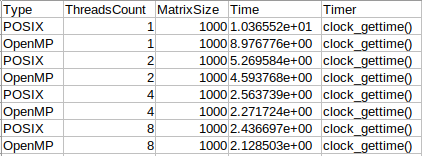


Таблица 1. Csv-файл c размерностью матрицы 1000x1000.

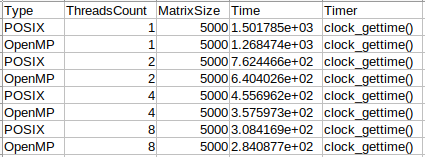


Таблица 2. Csv-файл c размерностью матрицы 5000x5000.

OpenMP работает быстрее поскольку, имеет преимущества кроссплатформенности и упрощения некоторых операций. Он обрабатывает многопоточность по-другому, так как предоставляет параметры многопоточности более высокого уровня, такие как распараллеливание циклов. POSIX Threads - это API нижнего уровня для явного создания потоков и синхронизации - в этом отношении он обеспечивает больший контроль.

**Приложение**

*main.cpp*

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <ctype.h>

#include <string.h>

#include <stdint.h>

#include <pthread.h>

#include <omp.h>

**struct** DGEMMArgs {

**double** \*\*matrixA;

**double** \*\*matrixB;

**double** \*\*matrixRes;

**long** **long** matrixSize;

**long** **long** from;

**long** **long** to;

};

**int** **getParameters**(**int** argc, **char** \*argv[], **long** **long** &matrixSize,**long** **long** &threadsCount){

**for** (**int** i = **1** ; i < argc ; i++)

{

**if** (strcmp("-s", argv[i]) == **0** || strcmp("--matrix-size", argv[i]) == **0**)

{

i++;

**size\_t** len = strlen(argv[i]);

**for** (**size\_t** j = **0** ; j < len; j++)

**if** (!isdigit(argv[i][j])){

printf("Error in arguments: invalid value for --matrix-size!**\n**The value must be a number!**\n**");

**return** **1**;

}

matrixSize = atoll(argv[i]);

}

**else** **if** (strcmp("-t", argv[i]) == **0** || strcmp("--threads-count", argv[i]) == **0**)

{

i++;

**size\_t** len = strlen(argv[i]);

**for** (**size\_t** j = **0** ; j < len ; j++)

**if** (!isdigit(argv[i][j])){

printf("Error in arguments: invalid value for --threads-count!**\n**The value must be a number!**\n**");

**return** **1**;

}

threadsCount = atoll(argv[i]);

}

**else** {

printf("Error in arguments: unknown key **\"**%s**\"\n**", argv[i]);

**return** **1**;

}

}

**return** **0**;

}

**void** **printMatrix**(**double** \*\*matrix, **long** **long** size){

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

**for** (**long** **long** j = **0**; j < size; j++)

printf("%.6f ", matrix[i][j]);

printf("**\n**");

}

}

**void** \* **DGEMM\_BLASS\_ForThread**(**void** \*threadArgs){

DGEMMArgs \*arg = (**struct** DGEMMArgs\*) threadArgs;

**for** (**long** **long** i = arg->from; i < arg->to; i++)

**for** (**long** **long** j = **0**; j < arg->matrixSize; j++){

arg->matrixRes[i][j] = **0**;

**for** (**long** **long** k = **0**; k < arg->matrixSize; k++)

arg->matrixRes[i][j] += arg->matrixA[i][k] \* arg->matrixB[k][j];

}

**return** nullptr;

}

**void** **DGEMM\_BLASS\_openMP**(**double**\*\* matrixA, **double**\*\* matrixB, **double**\*\* matrixRes, **long** **long** matrixSize){

**long** **long** i, j, k;

#pragma omp parallel **for** **private**(i, j, k) shared (matrixA, matrixB, matrixRes)

**for** (i = **0**; i < matrixSize; ++i)

**for** (j = **0**; j < matrixSize; ++j) {

matrixRes[i][j] = **0**;

**for** (k = **0**; k < matrixSize; ++k)

matrixRes[i][j] += matrixA[i][k] \* matrixB[k][j];

}

}

**int** **outToCSV**(**char**\* type, **long** **long** threadsCount, **long** **long** matrixSize, **double** time){

**FILE** \*fp;

**if** (!(fp = fopen("output.csv", "a"))){

printf("Error: can't open/find output.csv**\n**");

**return** **1**;

}

// fprintf(fp, "Type;threadsCount;matrixSize;time;Timer;\n");

fprintf(fp, "%s;%lld;%lld;%e;%s;**\n**",

type,

threadsCount,

matrixSize,

time,

"clock\_gettime()");

fclose(fp);

**return** **0**;

}

**int** **main**(**int** argc, **char** \*argv[]) {

srand(time(**0**));

**struct** timespec mt1{}, mt2{};

**double** time;

**long** **long** matrixSize = **10**;

**long** **long** threadsCount = **1**;

**if** (getParameters(argc, argv, matrixSize, threadsCount))

**return** **1**;

printf("arguments:**\n**");

printf("matrixSize = %lld **\n**", matrixSize);

printf("threadsCount = %lld **\n**", threadsCount);

**double** \*\*matrixA = **new** **double**\*[matrixSize];

**double** \*\*matrixB = **new** **double**\*[matrixSize];

**double** \*\*matrixRes = **new** **double**\*[matrixSize];

**for** (**long** **long** i = **0**; i < matrixSize; i++) {

matrixA[i] = **new** **double**[matrixSize];

matrixB[i] = **new** **double**[matrixSize];

matrixRes[i] = **new** **double**[matrixSize];

**for** (**long** **long** j = **0**; j < matrixSize; j++) {

matrixA[i][j] = rand() / **10000** + (**double**)rand() / RAND\_MAX;

matrixB[i][j] = rand() / **10000** + (**double**)rand() / RAND\_MAX;

matrixRes[i][j] = **0**;

}

}

time = **0**;

DGEMMArgs\* argsThreads = **new** DGEMMArgs[threadsCount];

**pthread\_t** \*threads = **new** **pthread\_t**[threadsCount];

**int** status = **0**;

**for** (**long** **long** i = **0**; i < threadsCount; i++) {

argsThreads[i].matrixA = matrixA;

argsThreads[i].matrixB = matrixB;

argsThreads[i].matrixRes = matrixRes;

argsThreads[i].matrixSize = matrixSize;

argsThreads[i].from = matrixSize / threadsCount \* i;

argsThreads[i].to = matrixSize / threadsCount \* (i + **1**);

}

clock\_gettime(CLOCK\_REALTIME, &mt1);

**for** (**long** **long** i = **0**; i < threadsCount; i++)

**if** (pthread\_create(&threads[i], NULL, DGEMM\_BLASS\_ForThread, &argsThreads[i])) {

printf("Error: Creating thread #%lld", i);

**return** **1**;

}

**for** (**long** **long** i = **0**; i < threadsCount; i++)

status += pthread\_join(threads[i], nullptr);

**if** (status != **0**)

printf("Error: Matrix multiplication failed (POSIX) **\n**");

clock\_gettime(CLOCK\_REALTIME, &mt2);

time = (**double**) (mt2.tv\_sec - mt1.tv\_sec) + (**double**) (mt2.tv\_nsec - mt1.tv\_nsec) / **1e9**;

outToCSV((**char**\*)"POSIX", threadsCount, matrixSize, time);

time = **0**;

omp\_set\_dynamic(**0**);

omp\_set\_num\_threads(threadsCount);

clock\_gettime(CLOCK\_REALTIME, &mt1);

DGEMM\_BLASS\_openMP(matrixA, matrixB ,matrixRes, matrixSize);

clock\_gettime(CLOCK\_REALTIME, &mt2);

time = (**double**) (mt2.tv\_sec - mt1.tv\_sec) + (**double**) (mt2.tv\_nsec - mt1.tv\_nsec) / **1e9**;

outToCSV((**char**\*)"OpenMP", threadsCount, matrixSize, time);

**for** (**long** **long** i = **0**; i < matrixSize; i++) {

**delete**(matrixA[i]);

**delete**(matrixB[i]);

**delete**(matrixRes[i]);

}

**delete**[](matrixA);

**delete**[](matrixB);

**delete**[](matrixRes);

**return** **0**;

}

*threads\_s.sh*

#!/bin/bash

echo "Type;ThreadsCount;MatrixSize;Time;Timer;" > ./output.csv

make

size=5000

from=1

to=8

step=2

**for** (( i=$from; i <= $to; i \*= $step ))

**do**

./a.out -s $size -t $i

**done**

filename=**$(**echo output\_$size**\\_**$from**\\_**$to**\\_**$step**)**

echo $filename

mv ./output.csv ./$filename.csv

*Makefile*

**all**:

g++ -Wall -Wextra -g -pthread -fopenmp \*.cpp