Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра вычислительных систем

Отчет по лабораторной работе

по дисциплине «Архитектура вычислительных систем»

Лабораторная работа №5

«Многопоточное программирование»

Выполнил: студент 3 курса
группы ИП-811
Мироненко К. А

Проверил: доцент кафедры ВС
Ефимов А. В.

Новосибирск 2020-21 уч. год

Оглавление

1. Постановка задачи

Тема: многопоточное программирование.

Задание: 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.

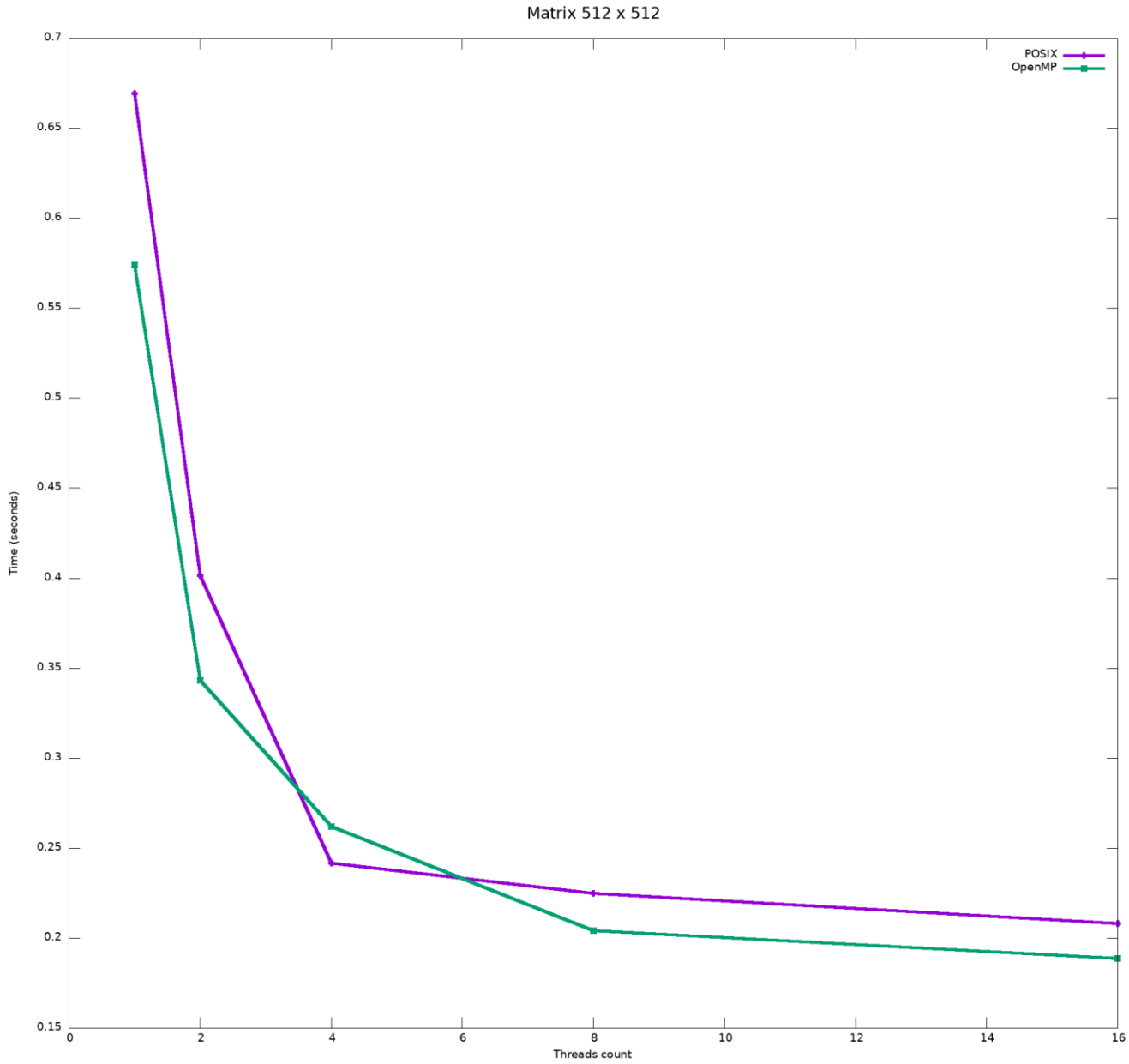
1) С использованием библиотеки стандарта POSIX Threads.
2) С использованием библиотеки стандарта OpenMP. 3) * С использованием библиотеки Intel TBB.
4) ** С использованием библиотеки стандарта MPI. Все матрицы помещаются в общей памяти одного вычислителя.
5) *** С использованием технологий многопоточности для графических сопроцессоров (GPU) — CUDA/OpenCL/OpenGL/OpenACC.

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

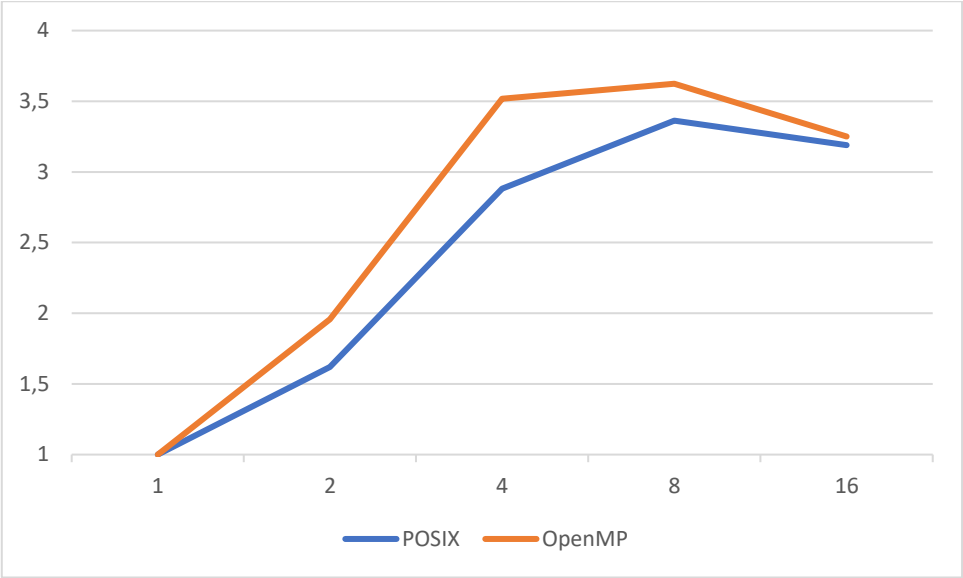
2. Примеры работы программы

	1	2	3	4	5	6
1	Type	ThreadsCo	MatrixSize	Time	Timer	
2	POSIX	1	1024	6.776913e+00	clock_gettime()	
3	OpenMP	1	1024	5.769160e+00	clock_gettime()	
4	POSIX	2	1024	3.338245e+00	clock_gettime()	
5	OpenMP	2	1024	2.904654e+00	clock_gettime()	
6	POSIX	4	1024	1.864884e+00	clock_gettime()	
7	OpenMP	4	1024	1.576318e+00	clock_gettime()	
8	POSIX	8	1024	2.202109e+00	clock_gettime()	
9	OpenMP	8	1024	1.928427e+00	clock_gettime()	
10	POSIX	16	1024	2.138315e+00	clock_gettime()	
11	OpenMP	16	1024	2.098875e+00	clock_gettime()	
12						
13						

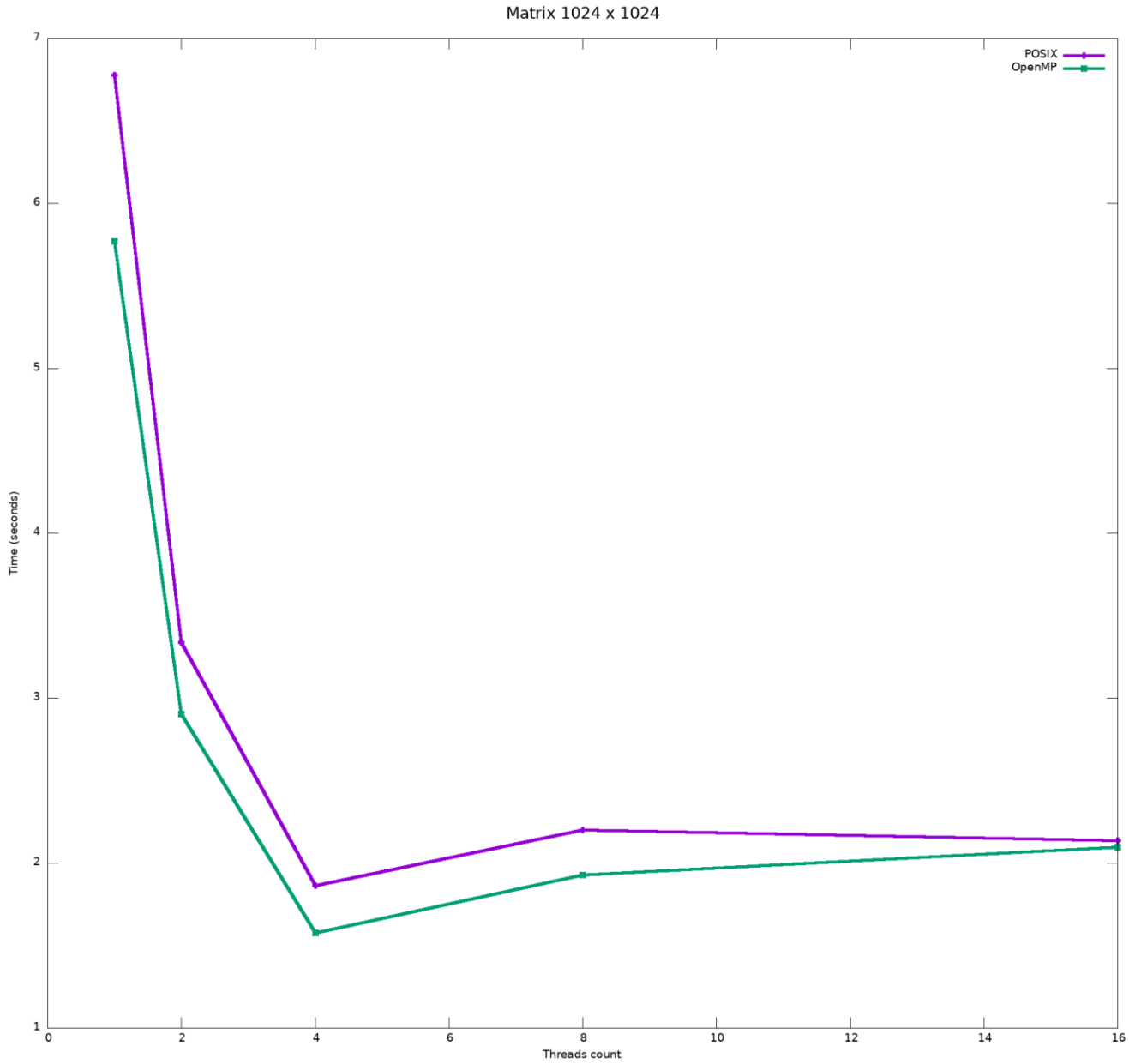
(csv файл результата работы)



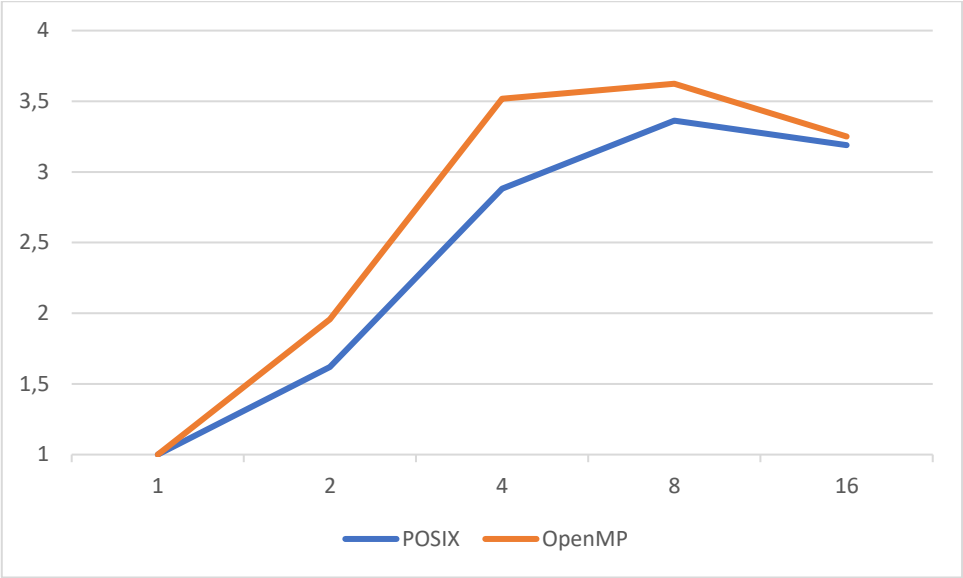
(График зависимости скорости выполнения от количества потоков при размере матриц 512)



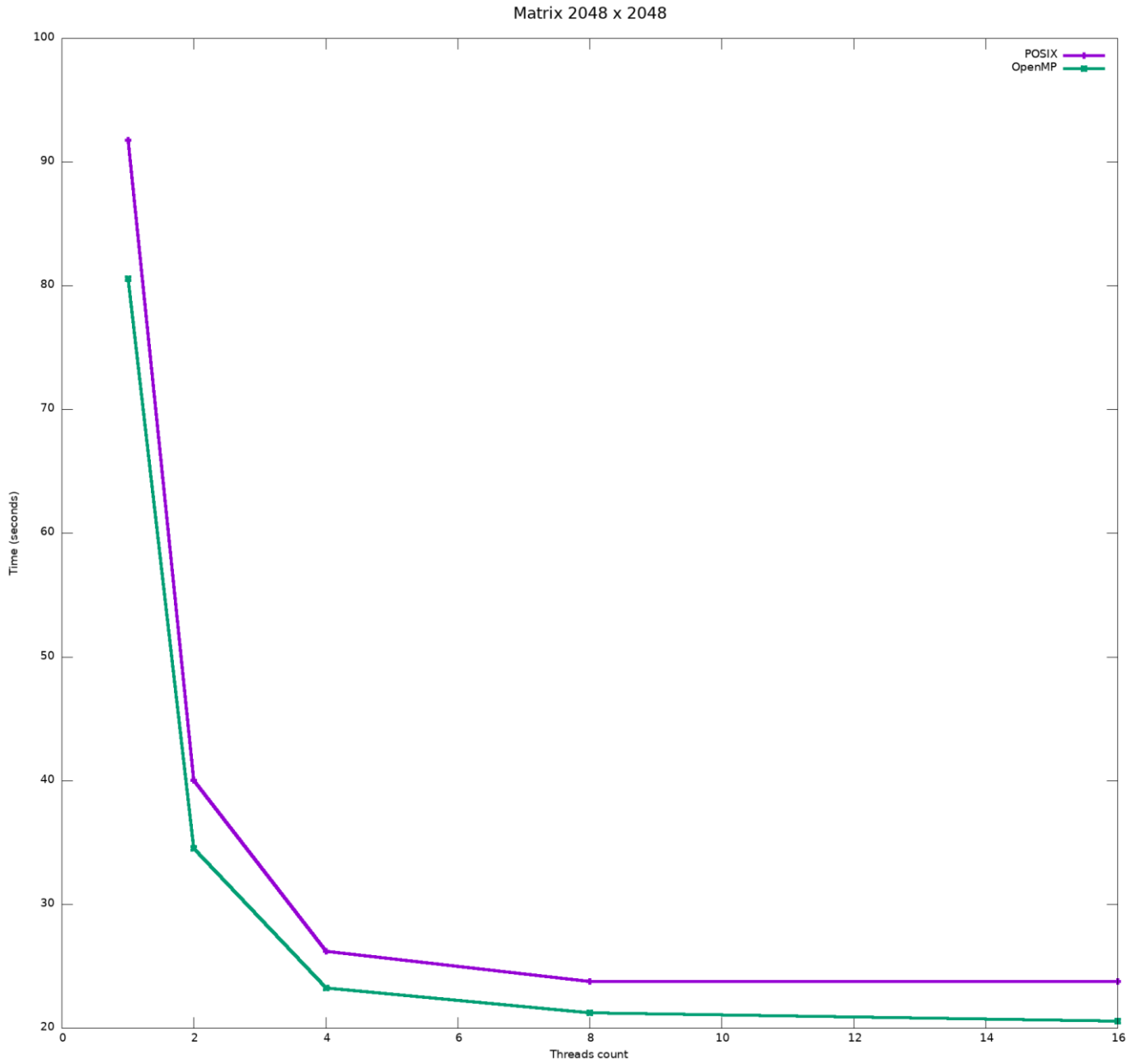
(График зависимости коэффициента ускорения многопоточной программы от числа потоков при размере матриц 512)



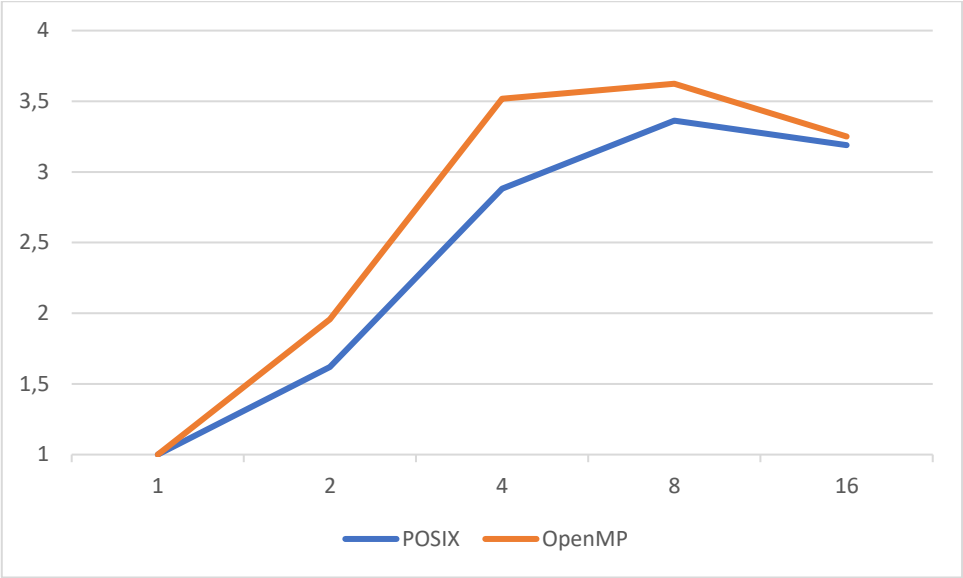
(График зависимости скорости выполнения от количества потоков при размере матриц 1024)



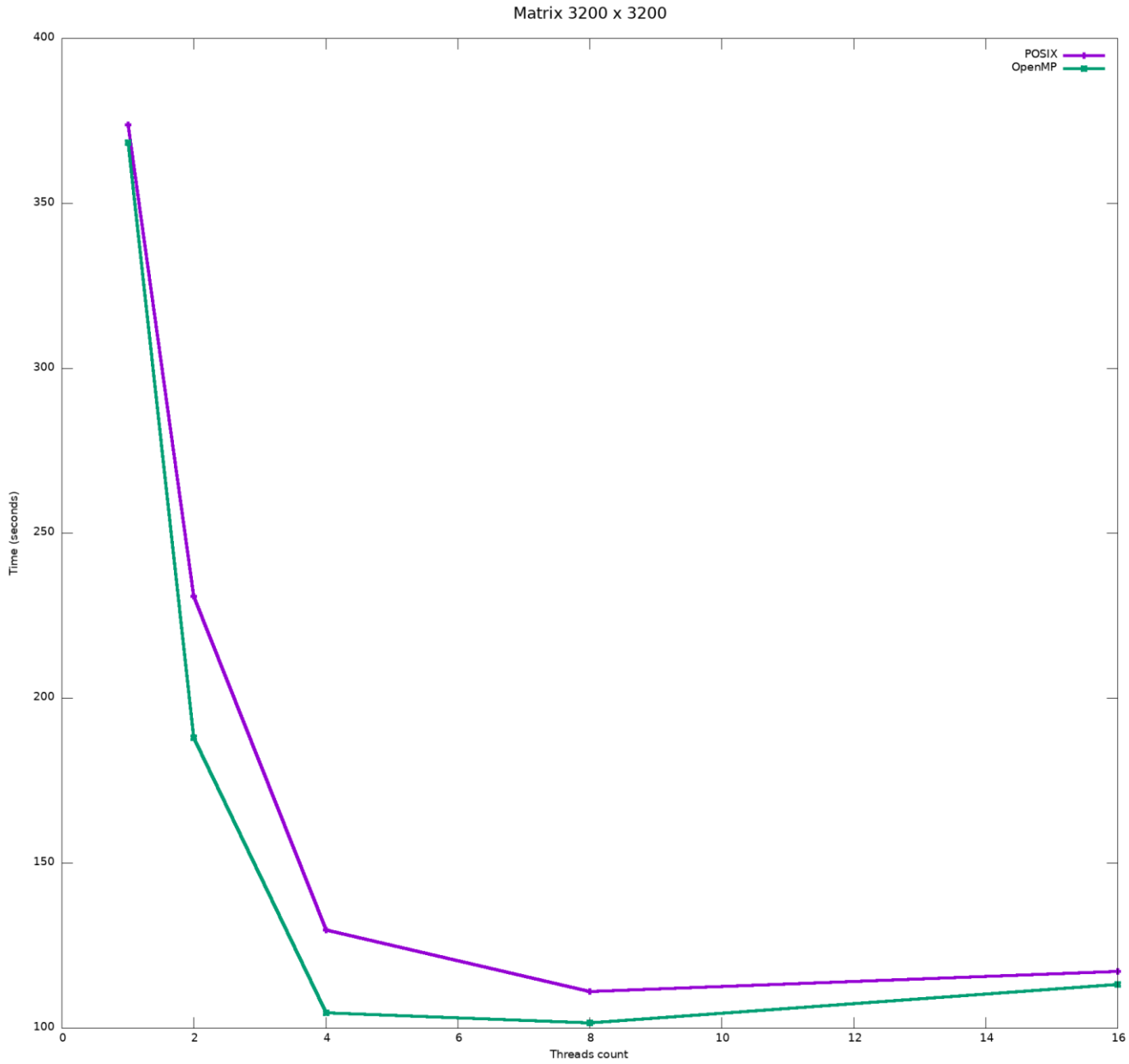
(График зависимости коэффициента ускорения многопоточной программы от числа потоков при размере матриц 1024)



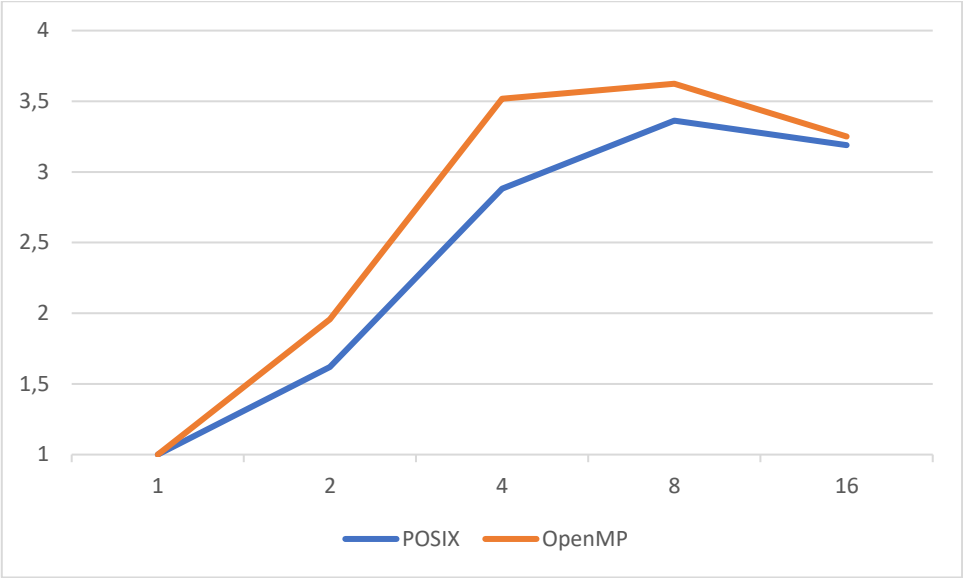
(График зависимости скорости выполнения от количества потоков при размере матриц 2048)



(График зависимости коэффициента ускорения многопоточной программы от числа потоков при размере матриц 2048)



(График зависимости скорости выполнения от количества потоков при размере матриц 3200)



(График зависимости коэффициента ускорения многопоточной программы от числа потоков при размере матриц 3200)

Приложение Листинг

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!\nThe 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!\nThe 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;
}
```