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. ИВ-823
Шиндель Э.Д.

Проверил: ст. преп. Кафедры ВС
Токмашева Е.И.

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.

2. 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.

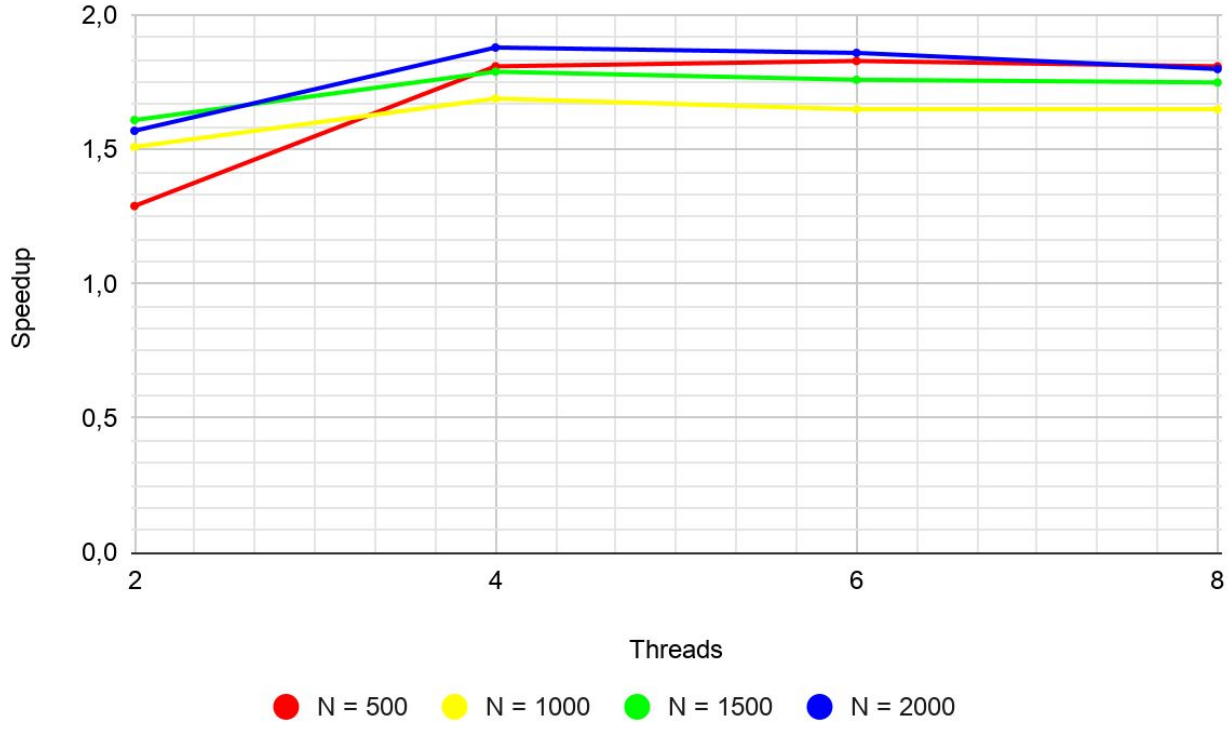
3. Определить оптимальное число потоков для вашего оборудования.

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

Результат работы

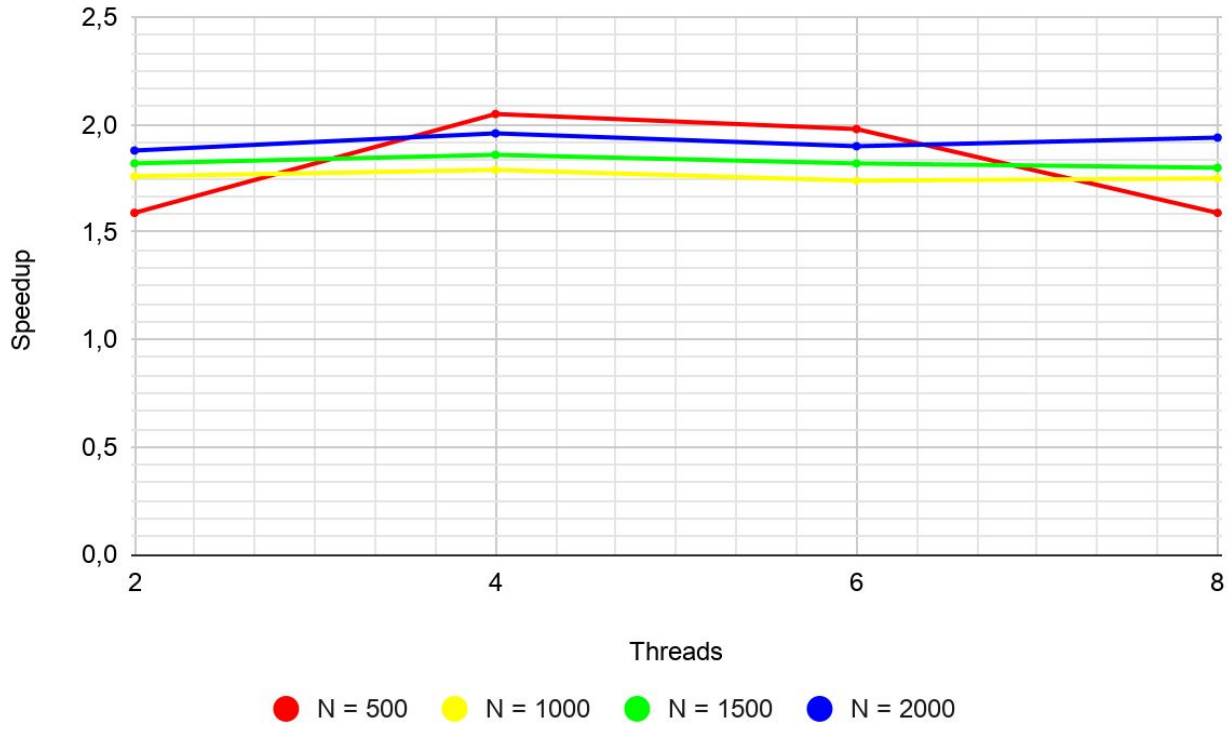
N	Threads								
	1	2		4		6		8	
	Time	Time	Speedup	Time	Speedup	Time	Speedup	Time	Speedup
500	1,21	0,94	1,29	0,67	1,81	0,66	1,83	0,67	1,81
1000	13,46	8,94	1,51	7,95	1,69	8,18	1,65	8,14	1,65
1500	56,00	34,89	1,61	31,25	1,79	31,83	1,76	32,09	1,75
2000	155,36	98,70	1,57	82,74	1,88	83,39	1,86	86,33	1,80

Многопоточность реализована с использованием библиотеки стандарта POSIX Threads



N	Threads								
	1	2		4		6		8	
	Time	Time	Speedup	Time	Speedup	Time	Speedup	Time	Speedup
500	1,21	0,94	1,29	0,67	1,81	0,66	1,83	0,67	1,81
1000	13,46	8,94	1,51	7,95	1,69	8,18	1,65	8,14	1,65
1500	56,00	34,89	1,61	31,25	1,79	31,83	1,76	32,09	1,75
2000	155,36	98,70	1,57	82,74	1,88	83,39	1,86	86,33	1,80

Многопоточность реализована с использованием библиотеки стандарта OpenMP



Вывод: исходя из результатов, приведённых на графиках, мы видим, что наилучшее ускорение достигается при 4-ёх потоках, но при 2-ух оно ближе всего к линейному, отсюда делаем вывод, что для моего устройства оптимальное число потоков равно 2.

Приложение

//pthreads.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>
#include <pthread.h>

typedef struct thread_info {
    pthread_t thread;
    int tid;
    int num_threads;
    int size;
    double **A;
    double **B;
    double **C;
} thread_info;

double wtime()
{
    struct timeval t;
    gettimeofday(&t, NULL);
    return (double)t.tv_sec + (double)t.tv_usec * 1E-6;
}

void *pthreads_dgemm_blas(void *arg) {
    thread_info *tinfo = arg;

    int start = tinfo->size / tinfo->num_threads * tinfo->tid;
    int end = (tinfo->tid + 1 != tinfo->size) ? (tinfo->size / tinfo->num_threads + start) :
tinfo->size;

    for (int i = start; i < end; i++) {
        for (int j = 0; j < tinfo->size; j++) {
            for (int k = 0; k < tinfo->size; k++) {
                tinfo->C[i][j] += tinfo->A[i][k] * tinfo->B[k][j];
            }
        }
    }
    pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
    int tmp, size, num_threads;
    if (argc != 3) {
        printf("./(prog_name) (size) (num_threads)\n");
        return 1;
    }
    size = atoi(argv[1]);
    num_threads = atoi(argv[2]);

    srand(time(NULL));
    thread_info tinfo[num_threads];

    double **A = (double **) malloc(sizeof(double *) * size);
    double **B = (double **) malloc(sizeof(double *) * size);
    double **C = (double **) malloc(sizeof(double *) * size);
    for (int i = 0; i < size; i++) {
        A[i] = (double *) malloc(sizeof(double) * size);
        B[i] = (double *) malloc(sizeof(double) * size);
        C[i] = (double *) malloc(sizeof(double) * size);
```

```
        for (int j = 0; j < size; j++) {
            A[i][j] = (double) (rand() % 201 - 100);
            B[i][j] = (double) (rand() % 201 - 100);
            C[i][j] = 0.0;
        }
    }

    for (int i = 0; i < num_threads; i++) {
        tinfo[i].tid = i;
        tinfo[i].num_threads = num_threads;
        tinfo[i].size = size;
        tinfo[i].A = A;
        tinfo[i].B = B;
        tinfo[i].C = C;
    }

    pthread_attr_t attr;
    pthread_attr_init(&attr);

    double t = wtime();
    for (int i = 0; i < num_threads; i++) {
        tmp = pthread_create(&tinfo[i].thread, &attr, pthreads_dgemm_blas, (void *)&tinfo[i]);
        if (tmp != 0) {
            printf("Creating thread %d is failed\n", i);
            return 1;
        }
    }

    for (int i = 0; i < num_threads; i++) {
        tmp = pthread_join(tinfo[i].thread, NULL);
        if (tmp != 0) {
            printf("Joining thread %d is failed", i);
            return 1;
        }
    }
    t = wtime() - t;

    printf("Time (%d threads) = %.2f sec\n", num_threads, t);
    free(A);
    free(B);
    free(C);
    return 0;
}
```

//omp.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "omp.h"

void omp_dgemm_blas(double **A, double **B, double **C, int size, int num_threads)
{
    omp_set_num_threads(num_threads);
    #pragma omp parallel for
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            for (int k = 0; k < size; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}

int main(int argc, char *argv[])
{
    int size, num_threads;
    if (argc != 3) {
        printf("./(prog_name) (size) (num_threads)\n");
        return 1;
    }
    size = atoi(argv[1]);
    num_threads = atoi(argv[2]);

    srand(time(NULL));

    double **A = (double **) malloc(sizeof(double *) * size);
    double **B = (double **) malloc(sizeof(double *) * size);
    double **C = (double **) malloc(sizeof(double *) * size);
    for (int i = 0; i < size; i++) {
        A[i] = (double *) malloc(sizeof(double) * size);
        B[i] = (double *) malloc(sizeof(double) * size);
        C[i] = (double *) malloc(sizeof(double) * size);
        for (int j = 0; j < size; j++) {
            A[i][j] = (double) (rand() % 201 - 100);
            B[i][j] = (double) (rand() % 201 - 100);
            C[i][j] = 0.0;
        }
    }

    double t = omp_get_wtime();
    omp_dgemm_blas(A, B, C, size, num_threads);
    t = omp_get_wtime() - t;

    printf("Time (%d threads) = %.2f sec\n", num_threads, t);
    free(A);
    free(B);
    free(C);
    return 0;
}
```