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 3](#_heading=h.gjdgxs)

[Результат работы 4](#_heading=h.1fob9te)

[Приложение 6](#_heading=h.3znysh7)

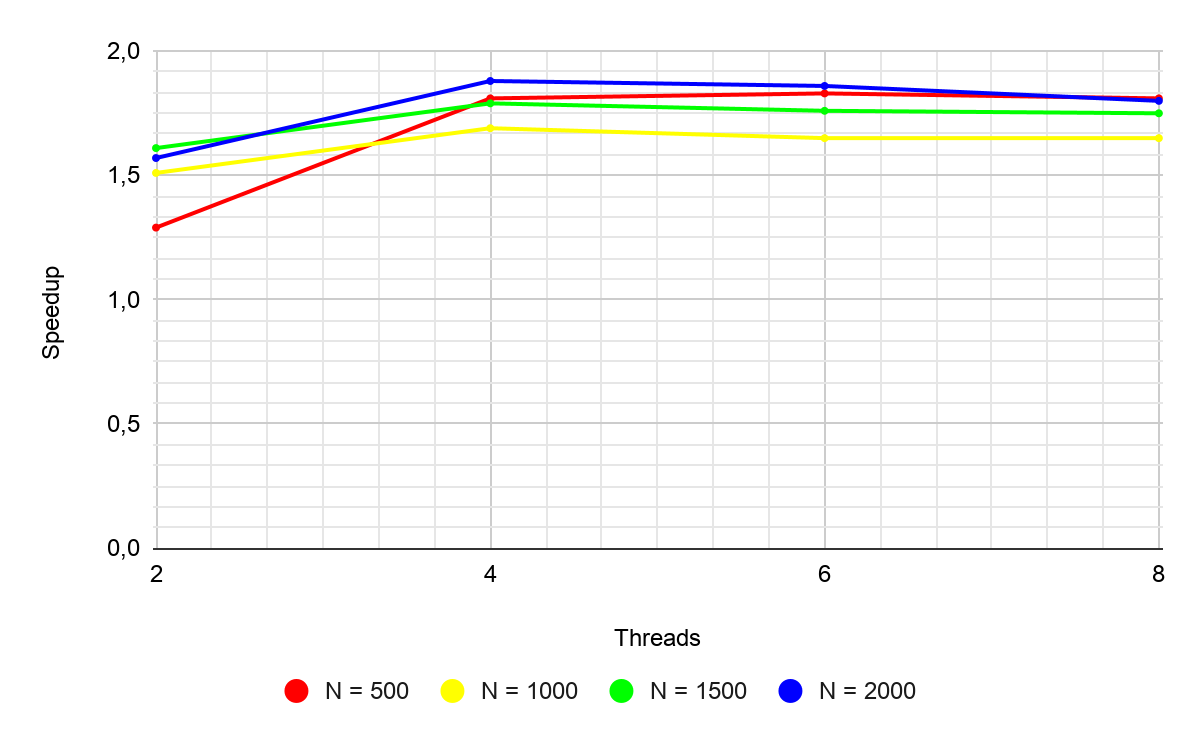
# Постановка задачи

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:
2. c использованием библиотеки стандарта POSIX Threads
3. c использованием библиотеки стандарта OpenMP.
4. 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.
5. Определить оптимальное число потоков для вашего оборудования.
6. Подготовить отчет отражающий суть, этапы и результаты проделанной работы.

# Результат работы

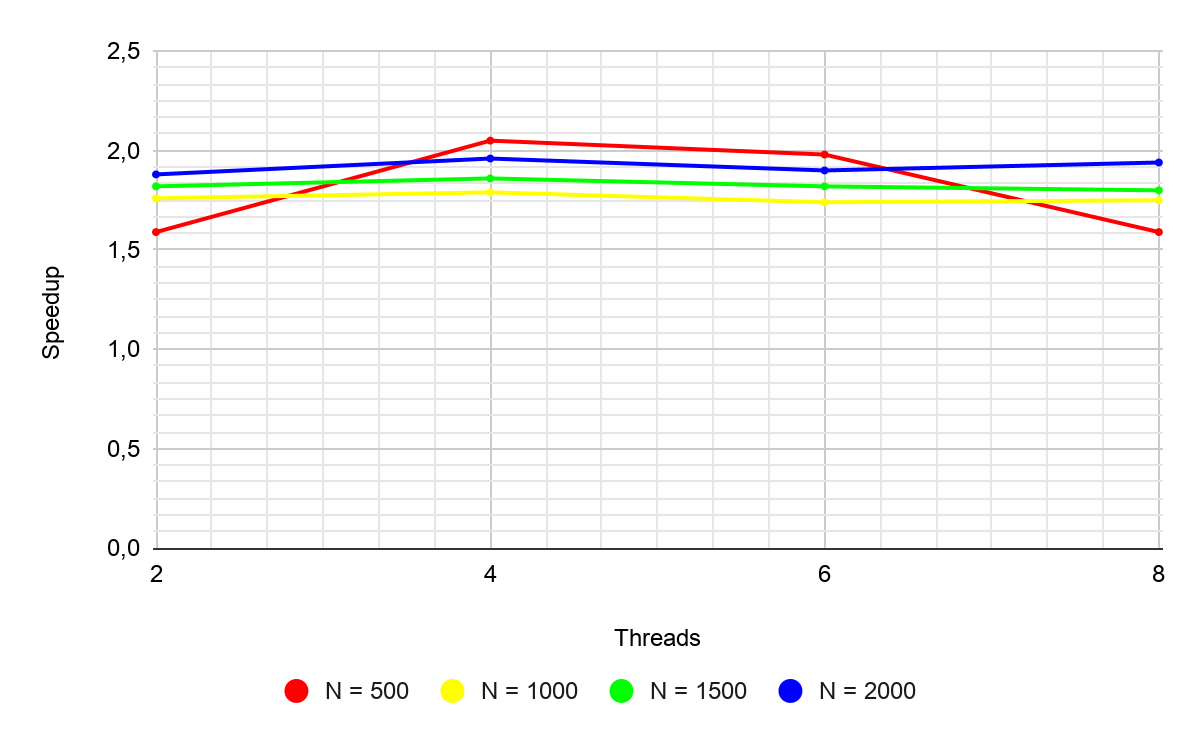
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | Threads | | | | | | | | |
| 1 | 2 | | 4 | | 6 | | 8 | |
| Time | Time | Speedup | Time | Speedup | Time | Speedup | Time | Speedup |
| 500 | 1,21 | 0,94 | 1,29 | 0,67 | 1,81 | 0,66 | 1,83 | 0,67 | 1,81 |
| 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;

}