Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 3

по курсу «Разработка параллельных и распределенных программ»

Решение систем линейных алгебраических уравнений итерационными методами с помощью библиотеки OpenMP

Студент: Пишикина М.В.

Группа: ИУ9-51Б

Преподаватель: Царев А.С.

Москва 2024

Содержание

1 Постановка задачи

Переписать из лабораторной работы №3 программу, которая реализует итерационный алгоритм решения системы линейных алгебраических уравнений вида Ax=b, исполтзуя библиотеку OpenMP

2 Практическая реализация

```
#include <math.h>
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

#define N 40000
#define EPSILON 1e-5
#define MAX_ITERATIONS 1500

// parallel  создаёт группу потоков, которые будут выполняться параллельно
void minimal_residual_method(double *A, double *b, double *x, int
 ↪  num_threads)
{
    double *y = (double *)malloc(N * sizeof(double));
    double *Ay = (double *)malloc(N * sizeof(double));

    for (int iter = 0; iter < MAX_ITERATIONS; iter++)
    {
// y = A * x - b : вычисление невязки
#pragma omp parallel for num_threads(num_threads)
        for (int i = 0; i < N; i++)
        {
            y[i] = 0.0;
            for (int j = 0; j < N; j++)
            {
```

```
                y[i] += A[i * N + j] * x[j];
            }
            y[i] -= b[i];
        }

        // норма невязки
        double residual_norm = 0.0;
#pragma omp parallel for reduction(+ : residual_norm)
 ↪    num_threads(num_threads)
        for (int i = 0; i < N; i++)
        {
            residual_norm += y[i] * y[i];
        }
        residual_norm = sqrt(residual_norm);

        if (residual_norm < EPSILON)
        {
           printf("Solution found after %d iterations with residual norm %.12f.\n",
                  iter + 1, residual_norm);
            break;
        }

// Ay = A * y
#pragma omp parallel for num_threads(num_threads)
        for (int i = 0; i < N; i++)
        {
            Ay[i] = 0.0;
            for (int j = 0; j < N; j++)
            {
                Ay[i] += A[i * N + j] * y[j];
            }
        }
```

```
        // tau_n = (y, Ay) / (Ay, Ay)
        double tau_numerator = 0.0, tau_denominator = 0.0;
#pragma omp parallel for reduction(+ : tau_numerator, tau_denominator) \
    num_threads(num_threads)
        for (int i = 0; i < N; i++)
        {
            tau_numerator += y[i] * Ay[i];
            tau_denominator += Ay[i] * Ay[i];
        }

        if (tau_denominator < 1e-12)
        {
            printf("Знаменатель для tau слишком мал\n");
            break;
        }

        double tau = tau_numerator / tau_denominator;

// x_(n+1) = x_n - tau * y : обновляем решение x
#pragma omp parallel for num_threads(num_threads)
        for (int i = 0; i < N; i++)
        {
            x[i] -= tau * y[i];
        }
    }

    free(y);
    free(Ay);
}

int main()
{
```

```
    double *A = (double *)malloc(N * N * sizeof(double)); // матрица
    ↪ коэффициентов
    double *b = (double *)malloc(N * sizeof(double));     // вектор свободныз
    ↪ членов
    double *x = (double *)malloc(N * sizeof(double));     // текущий вектор
    ↪ решения

#pragma omp parallel for
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            A[i * N + j] = 1.0;
        }
        A[i * N + i] = 2.0; // диагональные элементы
        b[i] = N + 1.0;
        x[i] = 0.1;
    }

    int num_threads = 16; // количество потоков
    omp_set_num_threads(num_threads);

    printf("Используется %d потоков.\n", num_threads);

    double start_time = omp_get_wtime();
    minimal_residual_method(A, b, x, num_threads);
    double end_time = omp_get_wtime();

    printf("Время выполнения: %.6f секунд\n", end_time - start_time);
    printf("Первые 10 элементов x: ");
    for (int i = 0; i < 10; i++)
    {
        printf("%.12f ", x[i]);
```

```
    }
    printf("\n");

    int valid = 1;
    for (int i = 0; i < N; i++)
    {
        if (fabs(x[i] - 1.0) > EPSILON)
        {
            valid = 0;
            break;
        }
    }
    printf("Действительное решение: %s\n", valid ? "true" : "false");

    free(A);
    free(b);
    free(x);

    return 0;
```

3 Характеристика устройства

Устройство: MacBook Pro 2020

Операционная система: macOS Sonoma

Процессор: Intel Core i5

Характеристика процессора: 4-ядерный чип, частота 2 ГГц

Оперативная память: 16GB LPDDR4X

4 Время работы программы

Взято N = 30000 , чтобы протестировать работу программы более точно.

Время работы при 1 поток: 54.848927

Время работы при 2 поток: 34.171449

Время работы при 3 поток: 27.469479

Время работы при 4 поток: 26.5138931

Время работы при 16 поток: 25.834463

Время работы при 32 поток: 24.351797

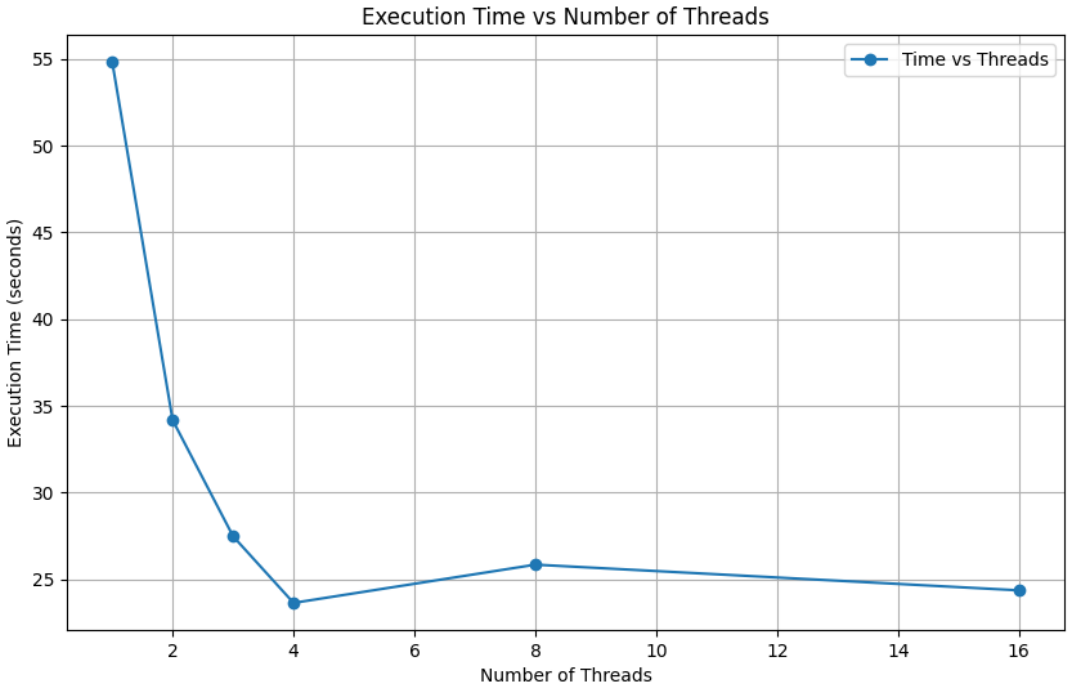

Рис. 1 — График: Зависимость времени выполнения от количества процессоров