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)» (МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»

Кафедра «Теоретическая информатика и компьютерные технологии»

Отчёт о лабораторной работе № 3 по курсу

«Разработка параллельных и

распределенных программ»

Параллельная реализация решения системы линейных алгебраических уравнений с помощью OpenMP

Студент: Белецкий В. С.

Группа: ИУ9-51Б

Преподаватель: Царев А.С.

Москва, 2022

Содержание

Постановка задачи

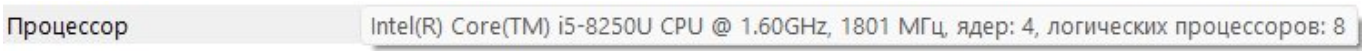
Пусть есть система из N линейных алгебраических уравнений в виде Ax=b, где A – матрица коэффициентов уравнений размером N×N, b – вектор правых частей размером N, x – искомый вектор решений размером N. Решение системы уравнений итерационным методом состоит в выполнении следующих шагов.

1. Задается x 0 – произвольное начальное приближение решения (вектор с произвольными начальными значениями).
2. Приближение многократно улучшается с использованием формулы вида x n+1 = f(x n), где функция f определяется используемым методом 1 .
3. Процесс продолжается, пока не выполнится условие g(x n) < ε, где функция g определяется используемым методом, а величина ε задает требуемую точность.

Выполнять при использовании библиотеки OpenMP.

Практическая реализация

Лабораторная работа выполнялась на ноутбуке с установленной ОС Windows 11, процессор имеет характеристики, указанные на скриншоте:



Для реализации поставленной задачи использовался язык программирования C.

Была создана программа, запускаемая в консоли при помощи bash-скрипта, который запускает программу на разном количестве потоков и замеряет время выполнения.

Ниже приведен листинг кода программы:

lab3.c:

```
#include <stdio.h>

#include <omp.h>
```

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

#define N 512

void matrixByVectorMultiply(double** matrix, double* vector, double* result) {
    int i, j;
#pragma omp parallel for shared(matrix, vector, result) private(i, j)
    for (i = 0; i < N; i++) {
        result[i] = 0;
        for (j = 0; j < N; j++)
            result[i] += (matrix[i][j] * vector[j]);
    }
}

void multVectorByScalar(double* vector, double scalar) {
    int i;
#pragma omp parallel for shared(vector) private(i)
    for (i = 0; i < N; i++)
        vector[i] *= scalar;
}

void diffVectors(double* self, double* vector) {
    int i;
#pragma omp parrallel for shared(self, vector) private(i)
    for (i = 0; i < N; i++)
        self[i] -= vector[i];
}

double vectorNorm(double* vector) {
```

```
    int i;
    double s = 0;
#pragma omp parrallel for shared(vector) private(i) reduction (+:s)
    for (i = 0; i < N; i++)
        s += (vector[i] * vector[i]);
    return sqrt(s);
}

int main() {
    double epsilon = 0.00001;
    double tau = 0.1 / N;

    double** A;
    A = (double**)malloc(sizeof(double*) * N);

    for (int i = 0; i < N; i++) {
        A[i] = (double*)malloc(sizeof(double) * N);
        for (int j = 0; j < N; j++) {
            if (i == j)
                A[i][j] = 2.0;
            else A[i][j] = 1.0;
        }
    }
    double* u;
    u = (double*)malloc(sizeof(double) * N);
    for (int i = 0; i < N; i++)
        u[i] = sin((2 * M_PI * i) / N);
    double* x;
    x = (double*)malloc(sizeof(double) * N);
    for (int i = 0; i < N; i++)
```

```
        x[i] = 0;
    double* b;
    double* y;
    b = (double*)malloc(sizeof(double) * N);
    y = (double*)malloc(sizeof(double) * N);
    matrixByVectorMultiply(A, u, b);
    double norm_b;
    norm_b = vectorNorm(b);
    while (1) {
        matrixByVectorMultiply(A, x, y);
        diffVectors(y, b);
        double norm_y;
        norm_y = vectorNorm(y);
        if ((norm_y / norm_b) < epsilon) {
            break;
        }
        multVectorByScalar(y, tau);
        diffVectors(x, y);
    }
    free(y);
    free(u);
    free(x);
    free(b);
    for (int i = 0; i < N; i++) {
        free(A[i]);
    }
    free(A);
    printf("Готово\n");
    return 0;
}
```

run.sh:

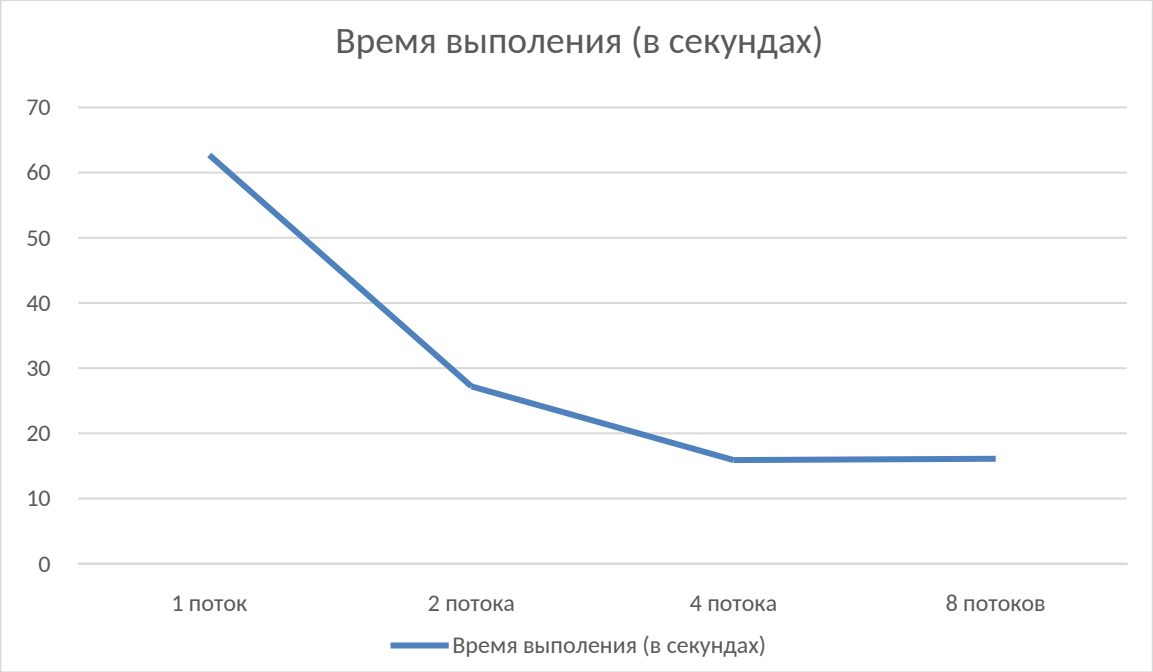
```
#!/bin/bash
export OMP_NUM_THREADS=1
TIMEFORMAT="Время выполнения %lR"
time {
   ./lab3
}

export OMP_NUM_THREADS=2
TIMEFORMAT="Время выполнения %lR"
time {
   ./lab3
}

export OMP_NUM_THREADS=4
TIMEFORMAT="Время выполнения %lR"
time {
   ./lab3
}

export OMP_NUM_THREADS=8
TIMEFORMAT="Время выполнения %lR"
time {
   ./lab3
}
```

Результаты

В ходе выполнения лабораторной работы были изучены способы распараллеливания вычислений с помощью потоков. Для каждой конфигурации исходных данных проводилось измерение затраченного время на вычисление. Результаты измерений прилагаются в виде таблицы и графика (указывается время

в секундах).

Количество потоков	Время
1	62.7
2	27.2
4	15.9
8	16.1



Выводы

Из приведённых данных видно, что использование потоков для вычисления произведения матриц позволило сократить время, затрачиваемое на вычисление. Также наглядно демонстрируется влияние количества потоков, на которых исполняется программа, на время выполнения.