Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа 2 по курсу

«Операционные системы»

III Семестр

Управление потоками. Обеспечение синхронизации между потоками.

Студент: Шляхтуров А.В
Группа: М8О-201Б-22
Преподаватель: Миронов Е. С.
Вариант 10
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2023

Цель работы.

Приобретение практических навыков в управлении процессами в ОС и обеспечении обмена данных между процессами посредством каналов

Задание

Решить систему линейных уравнений методом Гаусса

Решение

Программа может решать только систему уравнений, где количество переменных равно количеству уравнений и все уравнения линейно независимые. Такая система будет иметь единственное решение, которое можно найти с помощью метода Гаусса. Основная функция gauss() для каждой строки матрицы выполняет перемещение ведущего элемента в левый верхний угол матрицы, а затем деление каждой строки на ведущий элемент и вычитание первой строки из остальных, находящихся ниже.

В лабораторной работе реализован механизм пула потоков, когда несколько созданных потоков находятся в очереди, ожидая, когда можно будет забрать задачу, а во время решения системы несколько раз вызывается функция, которая подается потокам на выполнение. Потки один за другим выполняют функцию normalize() для разных строк матрицы. Эта функция выполняет деление одной выбранной строки на ведущий элемент матрицы.

Код

```
#include <vector>
#include <iostream>
#include <math.h>
#include <string>

#include <thread>
#include <mutex>
#include <condition_variable>
#include <atomic>
#include <chrono>
```

```
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

int taskCount = 0;
std::atomic<int> done_operation(0);

typedef struct Task
{
    int (*taskFunction)(double **, double *, int, int, int, double);
    double **matrix;
    double *y_vector;
    int i, k, size;
    double eps;
} Task;

Task taskQueue[100000];
pthread_mutex_t mutexQueue;
pthread_cond_t condQueue;

void executeTask(Task *task)
{
    done_operation ++;
    task->taskFunction(task->matrix, task->y_vector, task->i, task->k, task->size,
task->eps);
}

void submitTask(Task task)
{
    pthread_mutex_lock(&mutexQueue);
    taskQueue[taskCount] = task;
    taskCount++;
    pthread_mutex_unlock(&mutexQueue);
    pthread_cond_signal(&condQueue);
}

void *startThread(void *arg)
{
    int size = *((int *)arg);
    int value = pow(size, 2);
    while(1)
    {
        if(done_operation >= value){
            // std::cout << "ВЫШЕЛ" << std::endl;
             break;
        }
```

```
        pthread_mutex_lock(&mutexQueue);
        Task task;
        while (taskCount == 0)
        {
            pthread_cond_wait(&condQueue, &mutexQueue);
        }

        task = taskQueue[0];
        int i;
        for (i = 0; i < taskCount - 1; i++)
        {
            taskQueue[i] = taskQueue[i + 1];
        }
        taskCount--;
        pthread_mutex_unlock(&mutexQueue);
        executeTask(&task);
        // std::cout << "DONE_OPERATION = " << done_operation << " " <<std::endl;
        // std::cout << "VALUE = " << value << " " <<std::endl;


    }
    // std::cout << "ВЫШЕЛ" << std::endl;
    return NULL;
}


// Вывод системы уравнений
void sysout(double **a, double *y, int n)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            std::cout << a[i][j] << "*x" << j;
            if (j < n - 1)
                std::cout << " + ";
        }
        std::cout << " = " << y[i] << std::endl;
    }

    return;
}



int normalize(double **matrix, double *y_vector, int i, int k, int size, double
eps)
{
    usleep(1);
    double temp = matrix[i][k];
    if (abs(temp) < eps)
```

```
        return 0; // для нулевого коэффициента пропустить

    if (i < k)
        return 0;

    for (int j = k; j < size; j++)
    {
        matrix[i][j] = matrix[i][j] / temp;
    }
    y_vector[i] = y_vector[i] / temp;

    if (i == k)
        return 0; // уравнение не вычитать само из себя

    for (int j = k; j < size; j++)
    {
        matrix[i][j] = matrix[i][j] - matrix[k][j];
    }
    y_vector[i] = y_vector[i] - y_vector[k];
    return 0;
}



double *gauss(double **matrix, double *y_vector, int size, int thread_num)
{
    double *x, max;
    int k, index;
    const double eps = 0.00001; // точность
    x = new double[size];
    k = 0;

    pthread_mutex_init(&mutexQueue, NULL);
    pthread_cond_init(&condQueue, NULL);
    pthread_t th[thread_num];


    for (int i = 0; i < thread_num; i++)
        {
            pthread_create(&th[i], NULL, &startThread, (void *)&size);
        }

    while (k < size)
    {
        // Поиск строки с максимальным a[i][k]
        max = abs(matrix[k][k]);
        index = k;
        for (int i = k + 1; i < size; i++)
        {
            if (abs(matrix[i][k]) > max)
            {
```

```
                max = abs(matrix[i][k]);
                index = i;
            }
        }
        // Перестановка строк
        if (max < eps)
        {
            std::cout << "Решение получить невозможно из-за нулевого столбца ";
            std::cout << index << " матрицы A" << std::endl;
            return 0;
        }
        for (int j = 0; j < size; j++)
        {
            double temp = matrix[k][j];
            matrix[k][j] = matrix[index][j];
            matrix[index][j] = temp;
        }
        double temp = y_vector[k];
        y_vector[k] = y_vector[index];
        y_vector[index] = temp;

        // Нормализация уравнений

        for (int i = 0; i < size; i++)
        {
            Task t = {
                .taskFunction = &normalize,
                .matrix = matrix,
                .y_vector = y_vector,
                .i = i,
                .k = k,
                .size = size,
                .eps = eps};
            submitTask(t);
        }
        k++;
    }

    for (int i = 0; i < thread_num; i++)
        {
            // std::cout << " ID END PROCESS "
            //    << " = " << std::this_thread::get_id() << std::endl;
            pthread_join(th[i], NULL);
        }
        pthread_mutex_destroy(&mutexQueue);
        pthread_cond_destroy(&condQueue);



    // обратная подстановка
```

```
    for (int k = size - 1; k >= 0; k--)
    {
        x[k] = y_vector[k];
        for (int i = 0; i < k; i++)
            y_vector[i] = y_vector[i] - matrix[i][k] * x[k];
    }

    return x;
}

int main()
{


    int thread_num;
    std::cout << "Введите количество потоков: ";
    std::cin >> thread_num;
    std::cout << std::endl;

    // НИЖЕ БЛОК КОДА ДЛЯ РУЧНОГО ЗАПОЛНЕНИЯ МАТРИЦЫ

    // double **matrix, *y_vector, *x;
    // int size;
    // std::cout << "Введите количество уравнений: ";
    // std::cin >> size;
    // matrix = new double *[size];
    // y_vector = new double[size];
    // for (int i = 0; i < size; i++)
    // {
    //     matrix[i] = new double[size];
    //     for (int j = 0; j < size; j++)
    //     {
    //         std::cout << "a[" << i << "][" << j << "]= ";
    //         std::cin >> matrix[i][j];
    //     }
    // }
    // for (int i = 0; i < size; i++)
    // {
    //     std::cout << "y[" << i << "]= ";
    //     std::cin >> y_vector[i];
    // }


    // Инициализируем матрицу и заполняем рандомными числами диагональ и игреки
    int size;
    std::cout << "Введите размер массива: ";
    std::cin >> size;
    std::cout << std::endl;
    double **matrix;
    double *y_vector;
    double *x;
```

```
y_vector = new double[size];
matrix = new double *[size];

for (int i = 0; i < size; i++)
{
    matrix[i] = new double[size];
    for (int j = 0; j < size; j++)
    {
        if (i == j)
        {
            // std::cout << "a["<< i <<"]["<< j <<"] = ";
            // std:: cin >> matrix[i][j];
            matrix[i][j] = rand() % (50000 - (-50000) + 1) + (-50000);
        }
        else
        {
            matrix[i][j] = 0;
        }
    }
}

for (int i = 0; i < size; i++)
{
    // std::cout << "y[" << i << "]= ";
    // std::cin >> y_vector[i];
    y_vector[i] = rand() % (50000 - (-50000) + 1) + (-50000);
}

//Выводим введенную матрицу
sysout(matrix, y_vector, size);
std::cout << std::endl;



// std::cout << "MAIN PROCESS ID " << std::this_thread::get_id() << std::endl;

//Начинаем замерять время
auto start = std::chrono::high_resolution_clock::now();


// Вызываем решающую функцию
x = gauss(matrix, y_vector, size, thread_num);


//Заканчиваем замерять время
auto end = std::chrono::high_resolution_clock::now();
std::chrono::duration<double> duration = end - start;
std::cout << "Time taken: " << duration.count() << " seconds" << std::endl;


//Выводим найденные иксы
```

```
    for (int i = 0; i < size; i++)
        std::cout << "x[" << i << "]=" << x[i] << std::endl;

    return 0;
}
```

Пример работы

```
Введите количество потоков: 5

Введите размер массива: 10

21341*x0 + 0*x1 + 0*x2 + 0*x3 + 0*x4 + 0*x5 + 0*x6 + 0*x7 + 0*x8 + 0*x9 = 42111
0*x0 + -27583*x1 + 0*x2 + 0*x3 + 0*x4 + 0*x5 + 0*x6 + 0*x7 + 0*x8 + 0*x9 = 26523
0*x0 + 0*x1 + 25961*x2 + 0*x3 + 0*x4 + 0*x5 + 0*x6 + 0*x7 + 0*x8 + 0*x9 = 10857
0*x0 + 0*x1 + 0*x2 + -30231*x3 + 0*x4 + 0*x5 + 0*x6 + 0*x7 + 0*x8 + 0*x9 = -40966
0*x0 + 0*x1 + 0*x2 + 0*x3 + -21784*x4 + 0*x5 + 0*x6 + 0*x7 + 0*x8 + 0*x9 = 27315
0*x0 + 0*x1 + 0*x2 + 0*x3 + 0*x4 + -15907*x5 + 0*x6 + 0*x7 + 0*x8 + 0*x9 = 44252
0*x0 + 0*x1 + 0*x2 + 0*x3 + 0*x4 + 0*x5 + 28188*x6 + 0*x7 + 0*x8 + 0*x9 = 16889
0*x0 + 0*x1 + 0*x2 + 0*x3 + 0*x4 + 0*x5 + 0*x6 + -6005*x7 + 0*x8 + 0*x9 = 18023
0*x0 + 0*x1 + 0*x2 + 0*x3 + 0*x4 + 0*x5 + 0*x6 + 0*x7 + -39316*x8 + 0*x9 = 36132
0*x0 + 0*x1 + 0*x2 + 0*x3 + 0*x4 + 0*x5 + 0*x6 + 0*x7 + 0*x8 + -20475*x9 = -7298

Time taken: 0.00170581 seconds
x[0]=1.97324
x[1]=-0.961571
x[2]=0.418204
x[3]=1.3551
x[4]=-1.2539
x[5]=-2.78192
x[6]=0.599156
x[7]=-3.00133
x[8]=-0.919015
x[9]=0.356435
```

Выводы

В этой лабораторной работе я реализовал алгоритм решения системы линейных уравнений методом Гаусса в многопоточном режиме. Я узнал про различные примитивы синхронизации, такие как семафоры и мьютексы, а так же про conditional variable, которую я использовал для работы с пулом потоков.

Используя многопоточные вычисления, мне удалось добиться практически линейного прироста ускорения при добавлении потоков до 6 штук, т. к. на моём комьютере процессор с шестью физическими ядрами.