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»

Лабораторная работа № 5_1

по курсу «Распределение параллельных и распределённых программ»

«Синхронизация потоков»

Студент группы ИУ9-51Б Горбунов А. Д.

Преподаватель Царёв А. С.

Москва 2024

1 Задача

Использование барьерной синхронизации - задача "эволюция". Дан двумерный массив клеток, каждая из которых либо содержит организм (1), либо пуста (0), изначально он заполняется случайными значениями. Каждая клетка проверяет состояние своих соседей (их 8) и изменяет своё по правилам:

- Живая клетка, вокруг которой < 2 живых клеток, умирает от одиночества.
- Живая клетка, вокруг которой есть 2 или 3 живых клеток, выживает.
- Живая клетка, вокруг которой > 3 живых клеток, умирает от перенаселения.
- Пустая клетка, рядом с которой равно 3 живых соседа, оживает.

Реализовать заданное количество шагов моделирования при помощи n потоков. Каждый поток должен вычислить значения в заданной ему полосе матрицы. На каждом шаге результат моделирования необходимо записывать в новую матрицу. По окончании очередного шага необходимо скопировать содержимое новой матрицы в исходную. Шаги между потоками синхронизировать с помощью барьера (ни один из потоков не должен начинать следующий шаг, пока все не закончили текущий). Учесть следующие моменты:

- у клеток, находящихся на первой строке, первом столбце, последней строке и последнем столбце, соседями являются клетки с противоположной стороны матрицы
- каждый поток видит только свою часть матрицы, поэтому если ему необходим элемент, которого в его части матрицы нет, он должен каким-либо образом составлять запрос на то, чтобы тот поток, в котором этот элемент есть, его ему предоставил.

Замерить среднее время выполнения одного шага алгоритма и сравнить со средним временем выполнения одного шага без использования потоков.

Результат внести в отчёт.

2 Код решения

Файл main.cpp:

```
#include <iostream>
#include <vector>
#include <thread>
#include <chrono>
#include <cstdlib>
#include <ctime>

using namespace std;

void initMatrix(vector<vector<int>>& matrix, int rows, int cols) {
    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
            matrix[i][j] = rand() % 2;
}

int countLiveNeighbors(const vector<vector<int>>& matrix, int rows, int cols, int x
    int count = 0;
    for (int i = -1; i <= 1; i++)
        for (int j = -1; j <= 1; j++) {
            if (i == 0 && j == 0) continue;
            int nx = (x + i + rows) % rows;
            int ny = (y + j + cols) % cols;
            count += matrix[nx][ny];
        }
    return count;
}

void evolveStep(const vector<vector<int>>& currentMatrix, vector<vector<int>>&
    for (int i = startRow; i < endRow; ++i)
        for (int j = 0; j < cols; ++j) {
```

```
            int liveNeighbors = countLiveNeighbors(currentMatrix, rows, cols, i, j);
            if (currentMatrix[i][j] == 1) {
                if (liveNeighbors < 2 || liveNeighbors > 3)
                    nextMatrix[i][j] = 0;

            } else
                if (liveNeighbors == 3)
                    nextMatrix[i][j] = 1;
        }
}

void evolveStepWithThreads(vector<vector<int>>& currentMatrix, vector<vector<
    vector<thread> threads;
    int chunkSize = rows / numThreads;

    for (int i = 0; i < numThreads; ++i) {
        int startRow = i * chunkSize;
        int endRow = (i == numThreads - 1) ? rows : (i + 1) * chunkSize;
        threads.emplace_back(evolveStep, ref(currentMatrix), ref(nextMatrix), rows, co
    }

    for (auto& t : threads)
        t.join();
}

void threadMain(vector<vector<int>>& currentMatrix, vector<vector<int>>& nex
    initMatrix(currentMatrix, rows, cols);

    auto start = chrono::high_resolution_clock::now();

    for (int step = 0; step < numSteps; ++step) {
        evolveStepWithThreads(currentMatrix, nextMatrix, rows, cols, numThreads);
        currentMatrix = nextMatrix;
```

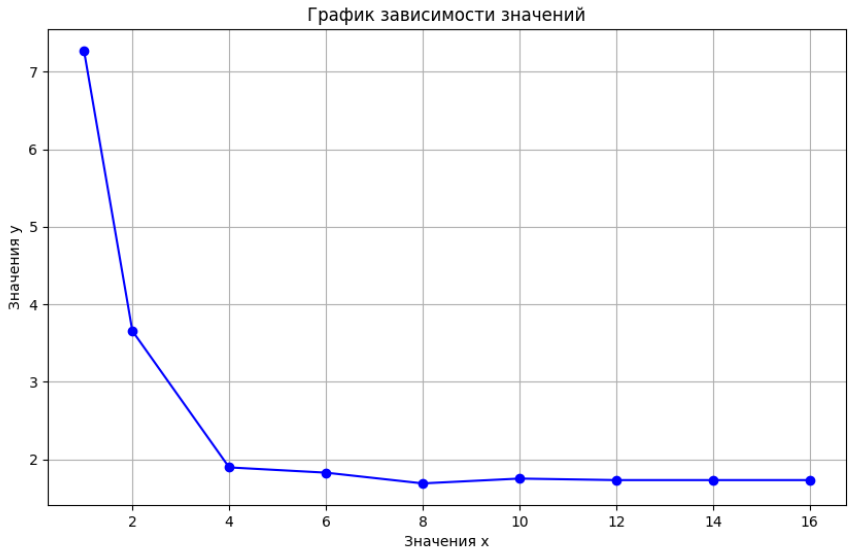
```
    }

    auto end = chrono::high_resolution_clock::now();
    chrono::duration<double> duration = end - start;
    cout << "Среднее время выполнения с "<< numThreads <<" потоками: " <<
}

int main() {
    srand(time(0));

    int rows = 10000;
    int cols = 10000;
    int numSteps = 100;

    vector<vector<int>> currentMatrix(rows, vector<int>(cols));
    vector<vector<int>> nextMatrix(rows, vector<int>(cols));

    threadMain(currentMatrix, nextMatrix, rows, cols, numSteps, 2);
    threadMain(currentMatrix, nextMatrix, rows, cols, numSteps, 4);
    threadMain(currentMatrix, nextMatrix, rows, cols, numSteps, 6);
    threadMain(currentMatrix, nextMatrix, rows, cols, numSteps, 8);
    threadMain(currentMatrix, nextMatrix, rows, cols, numSteps, 10);
    threadMain(currentMatrix, nextMatrix, rows, cols, numSteps, 12);
    threadMain(currentMatrix, nextMatrix, rows, cols, numSteps, 14);
    threadMain(currentMatrix, nextMatrix, rows, cols, numSteps, 16);

    initMatrix(currentMatrix, rows, cols);

    auto start = chrono::high_resolution_clock::now();

    for (int step = 0; step < numSteps; ++step) {
        evolveStep(currentMatrix, nextMatrix, rows, cols, 0, rows);
        currentMatrix = nextMatrix;
```

```
    }

    auto end = chrono::high_resolution_clock::now();
    chrono::duration<double> duration = end - start;
    cout << "Среднее ремя выполнения без потоков: " << duration.count() / num

    return 0;
}
```

3 Результат запуска

```
goarty@GoComp:~/Documents/paral_program/lab_5_1/src$ c++ -o main.o main.cpp
goarty@GoComp:~/Documents/paral_program/lab_5_1/src$ ./main.o
Среднее время выполнения с 2 потоками: 3.65546 секунд
Среднее время выполнения с 4 потоками: 1.89654 секунд
Среднее время выполнения с 6 потоками: 1.82893 секунд
Среднее время выполнения с 8 потоками: 1.6905 секунд
Среднее время выполнения с 10 потоками: 1.75271 секунд
Среднее время выполнения с 12 потоками: 1.7331 секунд
Среднее время выполнения с 14 потоками: 1.71816 секунд
Среднее время выполнения с 16 потоками: 1.69445 секунд
Среднее ремя выполнения без потоков: 7.26859 секунд
```

4 График



5 Заключение

Из результата работы программы с потоками и без с размером квадратной матрицы 10000 и количеством шагов моделирования равном 100 можно понять что использование потоков в разы ускаряют выполнение данной программы.