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

Лабораторная работа №2 по курсу
«Операционные системы»

Тема работы
“Потоки”

Студент: Ковриженков Дмитрий Олегович
Группа: М8О-203Б-23
Вариант:19

Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Постановка задачи

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение максимального количества потоков, работающих в один момент времени, должно быть задано ключом запуска вашей программы.
19. Дан массив координат (x, y). Пользователь вводит число кластеров. Проведите кластеризацию методом k-средних

Общие сведения

В данной программе реализован алгоритм кластеризации K-Means с использованием многопоточности. Пользователь вводит количество потоков, число точек и их координаты, а также желаемое количество кластеров.

Основной алгоритм выполняется в классе KMeans, где:

Инициализируются начальные центроиды кластеров случайным образом.

Точки распределяются по кластерам на основе минимального расстояния (distance).

Обновляются координаты центроидов до тех пор, пока они не стабилизируются или не будет достигнуто максимальное число итераций.

Распределение точек по кластерам выполняется параллельно с использованием POSIX threads (pthread).

Время выполнения замеряется с помощью класса Timer, который фиксирует время работы алгоритма.

Программа использует:

pthread_create() и pthread_join() для управления потоками.

std::vector для хранения точек и кластеров.

distance() для вычисления расстояний между точками.

rand() для случайного выбора начальных центроидов.

Вывод

Разработка этой программы позволила изучить принципы многопоточного программирования и алгоритмы кластеризации данных. Были освоены следующие концепции:

Использование POSIX threads для распараллеливания вычислений.

Применение метода K-Means для кластеризации данных.

Управление памятью и синхронизация потоков.

Оптимизация вычислений за счет многопоточности, что ускоряет обработку больших наборов данных.

Приложение
src/KMeans.cpp

```
#include "KMeans.h"
#include "Utils.h"
#include <pthread.h>
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>

KMeans::KMeans(int k, int maxThreads) : k(k), maxThreads(maxThreads) {
    centroids.resize(k);
    std::srand(static_cast<unsigned int>(std::time(0)));
}

void KMeans::initializeCentroids(std::vector<Point>& points) {
    for(int i = 0; i < k; ++i){
        centroids[i] = points[std::rand() % points.size()];
    }
}
```

```
void* KMeans::assignClusters(void* arg) {
    ThreadData* data = static_cast<ThreadData*>(arg);
    for (int i = data->start; i < data->end; ++i) {
        double minDist = distance((*data->points)[i], (*data->centroids)[0]);
        int bestCluster = 0;
        for (int j = 1; j < data->k; ++j) {
            double dist = distance((*data->points)[i], (*data->centroids)[j]);
            if (dist < minDist) {
                minDist = dist;
                bestCluster = j;
            }
        }
        (*data->points)[i].cluster = bestCluster;
    }
    return nullptr;
}

void KMeans::run(std::vector<Point>& points) {
    initializeCentroids(points);

    bool changed;
    int iterations = 0;
    const int maxIterations = 100;

    do {
        changed = false;

        int pointsPerThread = points.size() / maxThreads;
```

```
    std::vector<pthread_t> threads(maxThreads);
    std::vector<ThreadData> threadData(maxThreads);

    for (int i = 0; i < maxThreads; ++i) {
        int start = i * pointsPerThread;
        int end;
        if (i == maxThreads - 1) {
            end = points.size();
        } else {
            end = (i + 1) * pointsPerThread;
        }

        threadData[i] = { &points, &centroids, start, end, k };

        if (pthread_create(&threads[i], nullptr, assignClusters, &threadData[i]) != 0)
{
            std::cerr << "Ошибка при создании потока\n";
            exit(1);
        }
    }

    for (int i = 0; i < maxThreads; ++i) {
        pthread_join(threads[i], nullptr);
    }

    std::vector<int> count(k, 0);
    std::vector<double> sumX(k, 0.0), sumY(k, 0.0);

    for (const auto &point : points) {
```

```
            sumX[point.cluster] += point.x;
            sumY[point.cluster] += point.y;
            count[point.cluster]++;
        }

        for (int i = 0; i < k; ++i) {
            if (count[i] > 0) {
                Point newCentroid(sumX[i] / count[i], sumY[i] / count[i]);

                if (centroids[i].x != newCentroid.x || centroids[i].y != newCentroid.y) {
                    changed = true;
                }
                centroids[i] = newCentroid;
            }
        }

        iterations++;

    } while (changed && iterations < maxIterations);
}

void KMeans::printResults(const std::vector<Point>& points) {
    for (int i = 0; i < k; ++i) {
        std::cout << "Кластер " << i + 1 << " центр: (" << centroids[i].x << ", " <<
centroids[i].y << ")\n";
    }

    for (size_t i = 0; i < points.size(); ++i) {
```

```
        std::cout << "Точка " << i + 1 << " (" << points[i].x << ", " << points[i].y <<
") принадлежит кластеру " << points[i].cluster + 1 << "\n";
    }


    //std::cout << iterations << '\n';
}
```

src/ Timer.cpp

```
#include "Timer.h"

Timer::Timer(){
    start = std::chrono::high_resolution_clock::now();
}

Timer::~Timer(){
    std::chrono::duration<float> time = std::chrono::high_resolution_clock::now() -
start;
    std::cout << "Время выполнения: " << time.count() << " секунд\n";
}
```

src/ Utils.cpp

```
#include "Utils.h"
#include <cmath>

double distance(const Point &a, const Point &b) {
    return std::sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}
```

include/ KMeans.h

```
#ifndef KMEANS_H
#define KMEANS_H

#include <vector>
#include "Point.h"

class KMeans {
public:
    KMeans(int k, int maxThreads);
    void run(std::vector<Point>& points);
    void printResults(const std::vector<Point>& points);
private:
```

```
    int k;
    int maxThreads;
    std::vector<Point> centroids;

    struct ThreadData {
        std::vector<Point>* points;
        std::vector<Point>* centroids;
        int start;
        int end;
        int k;
    };

    static void* assignClusters(void* arg);
    void initializeCentroids(std::vector<Point>& points);
};

#endif
include/ Point.h
#ifndef POINT_H
#define POINT_H

struct Point {
    double x, y;
    int cluster = -1;

    Point(double x_val = 0.0, double y_val = 0.0) : x(x_val), y(y_val), cluster(-1) {}
};

#endif
include/ Timer.h
#ifndef TIMER_H
#define TIMER_H

#include <chrono>
#include <iostream>

class Timer {
public:
    Timer();
    ~Timer();
private:
    std::chrono::time_point<std::chrono::high_resolution_clock> start;
```

```
};

#endif
include/ Utils.h
#ifndef UTILS_H
#define UTILS_H

#include "Point.h"

double distance(const Point &a, const Point &b);

#endif
main.cpp
#include <iostream>
#include <vector>
#include "Point.h"
#include "KMeans.h"
#include "Timer.h"

int main() {
    int maxThreads;

    std::cout << "Введите максимальное количество потоков:\n";
    std::cin >> maxThreads;

    int n, k;
    std::cout << "Введите количество точек:\n";
    std::cin >> n;

    std::vector<Point> points(n);
    std::cout << "Введите координаты точек (x y):\n";
    for (int i = 0; i < n; ++i) {
        std::cin >> points[i].x >> points[i].y;
    }

    std::cout << "Введите количество кластеров:\n";
    std::cin >> k;

    Timer t;
    KMeans kmeans(k, maxThreads);
    kmeans.run(points);
    //kmeans.printResults(points);
```

```
    std::cout << "Используемое количество потоков: " << maxThreads << std::endl;

    return 0;
}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10)

project(lab2 CXX)

set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED True)

add_executable(lab2
    main.cpp
    src/KMeans.cpp
    src/Timer.cpp
    src/Utils.cpp
)

target_include_directories(lab2 PRIVATE include)

find_package(Threads REQUIRED)

target_link_libraries(lab2 PRIVATE Threads::Threads)
```

Пример вывода:

```
dimasic@Dimasic:~/Desktop/git/Labs/Osi_Labs/build/lab2$ ./lab2
Введите максимальное количество потоков:
2
Введите количество точек:
10
Введите координаты точек (x y):
1.0 2.0
1.5 1.8
5.0 8.0
8.0 8.0
1.0 0.6
9.0 11.0
8.0 2.0
10.0 2.0
9.0 3.0
6.0 1.0
Введите количество кластеров:
2
Кластер 1 центр: (7.85714, 5)
Кластер 2 центр: (1.16667, 1.46667)
Точка 1 (1, 2) принадлежит кластеру 2
Точка 2 (1.5, 1.8) принадлежит кластеру 2
Точка 3 (5, 8) принадлежит кластеру 1
Точка 4 (8, 8) принадлежит кластеру 1
Точка 5 (1, 0.6) принадлежит кластеру 2
Точка 6 (9, 11) принадлежит кластеру 1
Точка 7 (8, 2) принадлежит кластеру 1
Точка 8 (10, 2) принадлежит кластеру 1
Точка 9 (9, 3) принадлежит кластеру 1
Точка 10 (6, 1) принадлежит кластеру 1
Используемое количество потоков: 2
Время выполнения: 0.00103005 секунд
dimasic@Dimasic:~/Desktop/git/Labs/Osi_Labs/build/lab2$
```