МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Учреждение образование «Полоцкий государственный университет имени Евфросинии Полоцкой»

Факультет информационных технологий
Кафедра вычислительных систем и сетей

Лабораторная работа №1
по дисциплине: «Распределенные вычисления»
на тему: «Реализация алгоритма кластеризации методом К-средних на CPU.»

Выполнил — Студент гр. 23-ИТ-1
Вечерский М.И.

Проверил — Преподаватель кафедры ВСиС
Васильева Д.М.

Полоцк 2025

Вариант: 8

Цель работы: ознакомиться с алгоритмом кластеризации методом К-средних и реализовать данный алгоритм на CPU.

Описание проделанной работы: в ходе лабораторной работы был реализован и исследован алгоритм кластеризации k-means (метод k-средних) на языке C++ с использованием стандартной библиотеки. Целью работы было изучить принципы функционирования алгоритма, особенности его реализации и оценить производительность при обработке большого объёма данных.

Для эксперимента были заданы следующие параметры:

- количество точек - 80000;
- размерность пространства - 9;
- число кластеров - 10;
- максимальное число итераций - 100.

(update step) — для каждого кластера вычислялось среднее арифметическое координат всех точек, отнесенных к нему на предыдущем шаге; полученные значения становились новыми координатами соответствующих центроидов.

Указанные шаги последовательно повторялись в течение фиксированного числа итераций (100). Для пересчета положения центроидов была разработана специализированная функция, обеспечивающая суммирование координат точек каждого кластера с последующим делением на их количество.

С целью оценки эффективности реализации проводилось измерение времени выполнения алгоритма на центральном процессоре с использованием стандартной функции измерения времени. По завершении вычислений программа выводила суммарную длительность обработки всех итераций.

В результате выполнения лабораторной работы была получена работоспособная реализация алгоритма k-means, способная обрабатывать объемные наборы многомерных данных и демонстрирующая ключевые этапы процесса кластеризации. Практическая работа способствовала закреплению навыков оперирования многомерными данными, применения циклических конструкций, использования динамических структур данных и освоению базовых методик анализа производительности программного кода.

Листинг 1 – алгоритм k-средних

```
Point avg_point(const std::vector<Point>& points) {
    Point center;
    for (int i = 0; i < 9; ++i) {
        double sum = 0.0;
        for (const auto& point : points) {
            sum += point.coords[i];
        }
        center.coords[i] = sum / points.size();
    }
    return center;
}
```

```
double distance(const Point& point, const Point& center) {
    double dist = 0.0;

   for (auto&& [p_coord, c_coord] : std::views::zip(point.coords,
center.coords)) {
        dist += (p_coord - c_coord) * (p_coord - c_coord);
    }

    return dist;
}
std::vector<Point> load_pointers(const std::string& path_to_dir)
{
    std::vector<Point> points;

                        for     (const     auto&     entry     :
std::filesystem::directory_iterator(path_to_dir)) {
         if (entry.is_regular_file() && entry.path().extension()
== ".txt") {
            std::ifstream file(entry.path());

            std::string line;
            while (std::getline(file, line)) {
                std::istringstream iss(line);
                Point p;
                for (int i = 0; i < 9; ++i) {
                    iss >> p.coords[i];
                }
                points.push_back(p);
            }
        }
    }
    return points;
}

void  save_all_results(const  std::vector<Point>&  points,  const
std::vector<Point>& centers) {
    std::filesystem::create_directories("../Data");

    std::ofstream out_points("../Data/kmeans_result.txt");
for (const auto& p : points) {
         for (double coord : p.coords) out_points << coord << "
";
        out_points << p.cluster_id << "\n";
    }
    out_points.close();

    std::ofstream out_centers("../Data/kmeans_centers.txt");
    for (int i = 0; i < centers.size(); ++i) {
          for (double coord : centers[i].coords) out_centers <<
coord << " ";
        out_centers << i << "\n";
    }
```

```
 out_centers.close();

     std::cout << "Результаты и центры сохранены в ../Data/" <<
std::endl;
}
std::vector<Point> kmeans(std::vector<Point>& points) {
    std::array<Point, K> centers;

    std::uniform_int_distribution<> dis(0, points.size() - 1);
    std::mt19937 gen(std::random_device {}());

    for (int i = 0; i < K; ++i) {
        centers[i] = points[dis(gen)];
        centers[i].cluster_id = i;
    }

    for (int iter = 0; iter < 100; ++iter) {
         std::for_each(std::execution::par_unseq, points.begin(),
points.end(), [&](Point& p) {
                                              double   min_dist   =
std::numeric_limits<double>::max();
            int best_id = 0;
            for (int i = 0; i < K; ++i) {
                double d = distance(p, centers[i]);
                if (d < min_dist) {
                    min_dist = d;
                    best_id = i;
                }
            }
            p.cluster_id = best_id;
        });

        std::vector<std::array<double, 9>> new_sums(K, { 0.0 });
        std::vector<size_t> counts(K, 0);

        for (const auto& p : points) {
            counts[p.cluster_id]++;
            for (size_t i = 0; i < 9; ++i) {
new_sums[p.cluster_id][i] += p.coords[i];
            }
        }

        bool changed = false;
        for (int i = 0; i < K; ++i) {
            if (counts[i] > 0) {
                for (size_t j = 0; j < 9; ++j) {
                    double new_val = new_sums[i][j] / counts[i];
                     if (std::abs(centers[i].coords[j] - new_val)
> 1e-4)
                        changed = true;
                    centers[i].coords[j] = new_val;
                }
```

```
            }
        }

        if (iter % 10 == 0) {
            std::println("Итерация: {}", iter);
               std::for_each(centers.begin(), centers.end(), [&]
(Point& p) {
                      std::println("Центроид {} координаты: {}",
p.cluster_id, p.coords);
            });
        }

        if (!changed) {
            if (iter % 10 != 0) {
                std::println("Итерация: {}", iter);
                std::for_each(centers.begin(), centers.end(), [&]
(Point& p) {
                       std::println("Центроид {} координаты: {}",
p.cluster_id, p.coords);
                });
            }
            break;
        }
    }
    return { centers.begin(), centers.end() };
}

int main() {
    try {
        std::vector<Point> points = load_pointers("../Data");
        std::println("Классификация начала");
        auto start = std::chrono::high_resolution_clock::now();
        std::vector<Point> centers = kmeans(points);
        auto end = std::chrono::high_resolution_clock::now();

        std::cout << std::format(
            "Классификация завершена за {} сек\n",
             std::chrono::duration_cast<std::chrono::seconds>(end
- start).count());
      save_all_results(points, centers);
    } catch (const std::exception& e) {
        std::cerr << "Ошибка: " << e.what() << std::endl;
        return 1;
    } catch (...) {
        std::cerr << "Неизвестная ошибка" << std::endl;
        return 1;
    }

    return 0;
}
```

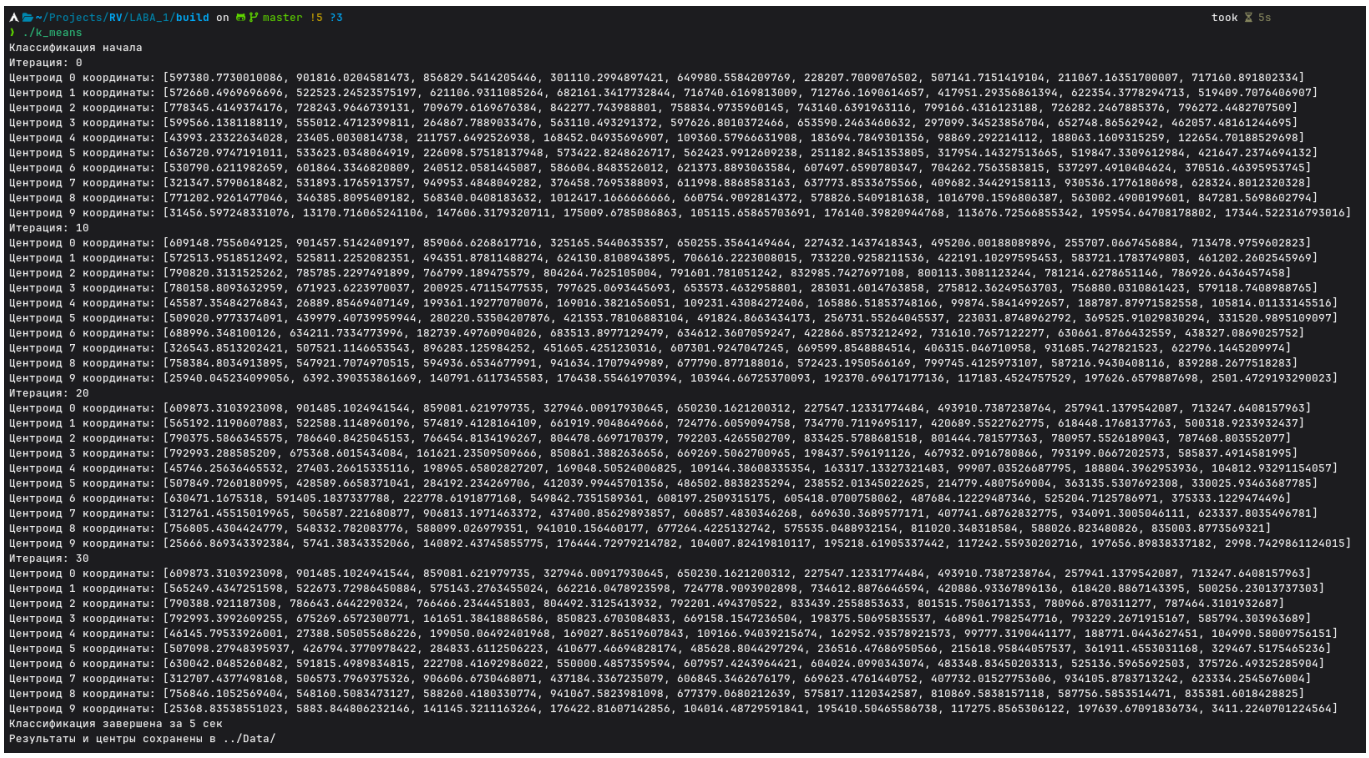

Рисунок 1 – результат кластеризации

Вывод: В ходе лабораторной работы была успешно реализована и исследована последовательная версия алгоритма кластеризации k-means, выполняемая на центральном процессоре. Практическая реализация позволила детально изучить архитектуру алгоритма, его базовые этапы и особенности обработки многомерных данных.

По итогам исследования были получены следующие результаты и выводы:

– Алгоритм k-means представляет собой итеративный процесс, состоящий из чередующихся шагов назначения точек кластерам и пересчета положения центроидов. Оба этапа характеризуются значительной вычислительной трудоемкостью при работе с большим количеством точек и кластеров.

– Анализ производительности CPU-реализации показал, что наибольшие временные затраты приходятся на вычисление расстояний между точками и центроидами, что соответствует теоретическим оценкам вычислительной сложности алгоритма.

– При обработке массива из 80 000 точек в 9-мерном пространстве алгоритм корректно выполнил заданное количество итераций, что подтвердило правильность реализации и её устойчивость в работе с объемными данными.

– Применение стандартных контейнеров и базовых структур данных языка C++ обеспечило читаемость и надежность кода, хотя и не позволило достичь предельно возможной производительности.

– Полученные результаты указывают на перспективность применения параллельных вычислений (с использованием технологий OpenMP, CUDA или SIMD-оптимизаций), оптимизации схем доступа к памяти, а также более

совершенных методов инициализации центроидов (например, k-means++) для ускорения работы алгоритма на крупномасштабных данных.

В целом, выполнение лабораторной работы способствовало углубленному освоению методов кластерного анализа, развитию навыков работы с многомерными данными и формированию представления о вычислительных затратах классических алгоритмов машинного обучения.