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 «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе № 4

по курсу «Анализ алгоритмов»

на тему: «Параллельные вычисления»

Студент ИУ7-52Б _____ А. П. Лемешев
(Группа) (Подпись, дата) (И. О. Фамилия)

Преподаватель _____ Ю. В. Строганов
(Подпись, дата) (И. О. Фамилия)

Преподаватель _____ Л. Л. Волкова
(Подпись, дата) (И. О. Фамилия)

Москва — 2022 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Применение параллельных вычислительных систем (ПВС) является стратегическим направлением развития вычислительной техники. Это обстоятельство вызвано не только принципиальным ограничением максимально возможного быстродействия обычных последовательных ЭВМ, но и практически постоянным наличием вычислительных задач, для решения которых возможностей существующих средств вычислительной техники всегда оказывается недостаточно [1].

Кластеризация — объединение в группы схожих объектов является одной из фундаментальных задач в области анализа данных. Список прикладных областей, где она применяется, широк: сегментация изображений, маркетинг, борьба с мошенничеством, прогнозирование, анализ текстов и многие другие [2].

В рамках данной лабораторной работы параллельные вычисления будут исследоваться на алгорите кластеризации DBSCAN.

1 Аналитическая часть

1.1 Цели и задачи

Цель работы: получить навыки организации многопоточных вычислений на примере реализации алгоритма DBSCAN.

Задачи лабораторной работы:

1) изучить понятия параллельных вычислений;
2) реализовать алгоритм DBSCAN с использованием параллельных вычислений;
3) провести сравнительный анализ времени работы алгоритма для различного количества используемых потоков;
4) обосновать полученные результаты в отчете.

1.2 Поток

Поток — это основная единица, которой операционная система выделяет время процессора. Каждый поток имеет приоритет планирования и набор структур, в которых система сохраняет контекст потока, когда выполнение потока приостановлено. Контекст потока содержит все сведения, позволяющие потоку безболезненно возобновить выполнение, в том числе набор регистров процессора и стек потока.

Выделяют 2 типа потоков: нативные и зеленые. Зелёные потоки — потоки, управление которыми производит виртуальная машина, нативные потоки — управление которыми производит операционная система.

Так называемые зеленые потоки лишь эмулируют многопоточную среду, не полагаясь на возможности операционной системы, что позволяет им работать без поддержки встроенных потоков. Однако зеленые потоки, как и любой инструмент имеет свои недостатки. Один из которых — блокировка всего процесса. Если на одном из потоков происходит блокировка, то блокируется весь процесс, ввиду того операционная система не знает о существовании зеленых потоков [3].

1.3 Алгоритм кластеризации DBSCAN

Алгоритм DBSCAN, плотностный алгоритм для кластеризации пространственных данных с присутствием шума, был предложен Мартином Эстер, Гансом-Питером Кригель и коллегами в 1996 году как решение проблемы разбиения данных на кластеры произвольной формы. Большинство алгоритмов, производящих плоское разбиение, создают кластеры по форме близкие к сферическим, так как минимизируют расстояние документов до центра кластера. Авторы DBSCAN экспериментально показали, что их алгоритм способен распознать кластеры различной формы, например, как на рисунке 1.1 [4, с. 197].

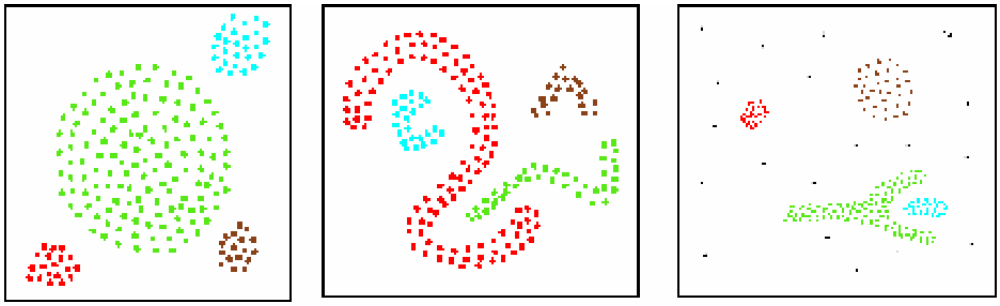

Рисунок 1.1 – Примеры кластеров произвольной формы

Идея, положенная в основу алгоритма, заключается в том, что внутри каждого кластера наблюдается типичная плотность точек, которая заметно выше, чем плотность снаружи кластера, а также плотность в областях с шумом ниже плотности любого из кластеров. Ещё точнее, что для каждой точки кластера её соседство заданного радиуса должно содержать не менее некоторого числа точек, это число точек задаётся пороговым значением. Перед изложением алгоритма дадим необходимые определения.

Eps-соседство точки p, обозначаемое как $N_{Eps}(p)$ определяется как множество документов, находящихся от точки p на расстояния не более Eps: $N_{Eps}(p) = \{q \in D | dist(p,q) \leq Eps\}$. Поиска точек, чьё $N_{Eps}(p)$ содержит хотя бы минимальное число точек ($MinPt$) не достаточно, так как точки бывают двух видов: ядровые и граничные.

Точка p *непосредственно плотно-достижима* из точки p (при заданных Eps и $MinPt$), если $p \in N_{Eps}(p)$ и $|N_{Eps}(p)| \geq MinPt$.

Точка p *плотно-достижима* из точки q (при заданных Eps и $MinPt$), если существует последовательность точек $q = p_1, p_2, \ldots, p_n = p : p_{i+1}$

непосредственно плотно-достижимы из p_i. Это отношение транзитивно, но не симметрично в общем случае, однако симметрично для двух ядровых точек.

Точка p *плотно-связана* с точкой q (при заданных Eps и $MinPt$), если существует точка o: p и q плотно-достижимы из o (при заданных Eps и $MinPt$).

Кластер C_j (при заданных Eps и $MinPt$) — это не пустое подмножество документов, удовлетворяющее следующих условиям:

1) $\forall p, q$: если $p \in C_j$ и q плотно-достижима из p (при заданных Eps и $MinPt$), то; $q \in C_j$;

2) $\forall p, q : p$ плотно-связана с q (при заданных Eps и $MinPt$).

Шум — это подмножество документов, которые не принадлежат ни одному кластеру: $\{p \in D | \forall j : p \notin C_j, j = \overline{1, |C|}\}$.

Алгоритм DBSCAN для заданных значений параметров Eps и $MinPt$ исследует кластер следующим образом: сначала выбирает случайную точку, являющуюся ядровой, в качестве затравки, затем помещает в кластер саму затравку и все точки, плотно-достижимые из неё.

Вывод

В данном разделе были рассмотрены понятия параллельных вычислений и алгоритм DBSCAN.

2 Конструкторская часть

2.1 Разработка алгоритмов

На рисунке 2.1 показана схема алгоритма DBSCAN.

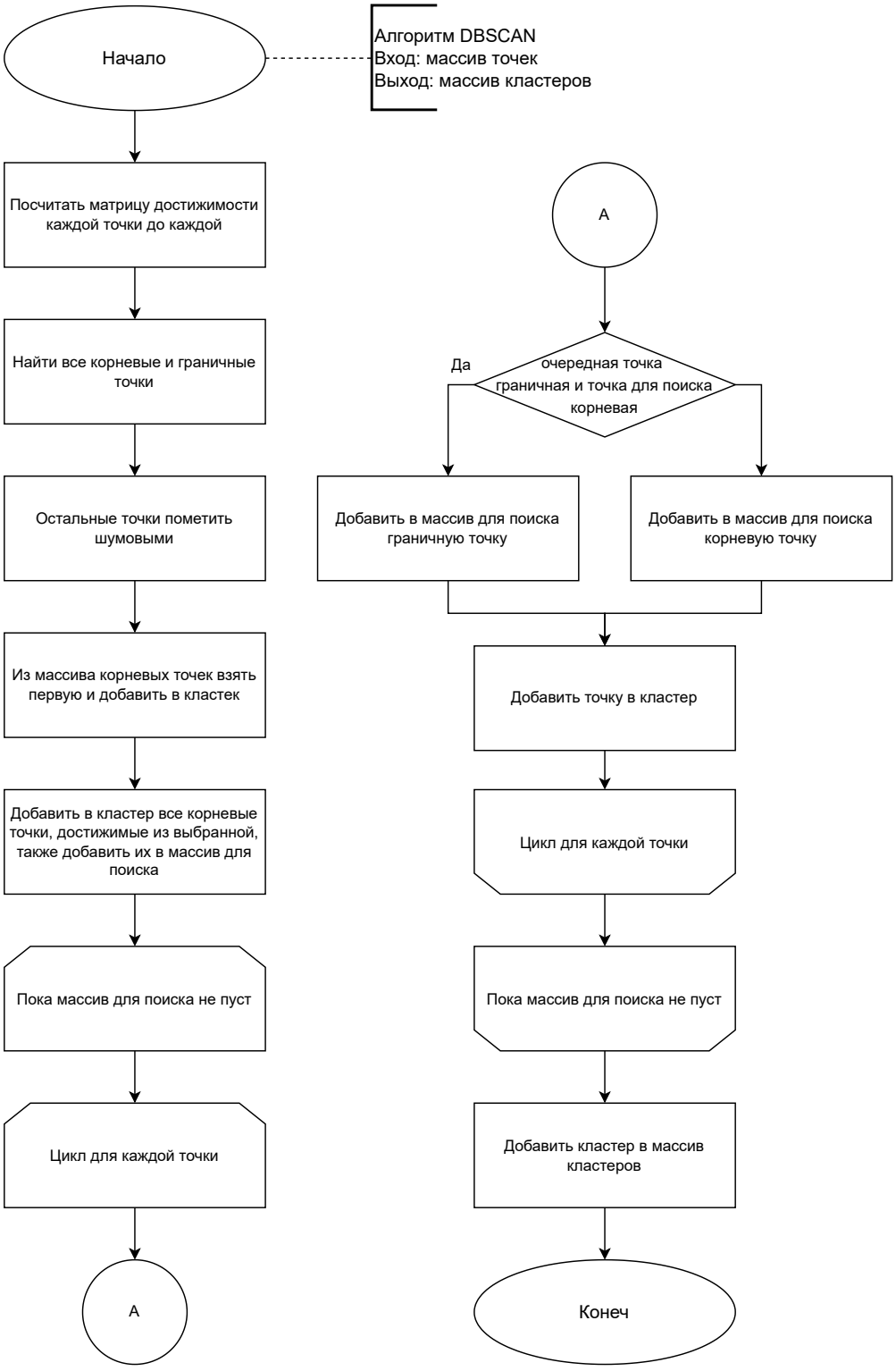

Рисунок 2.1 – Схема алгоритма DBSCAN

Вывод

На основе формул и теоретических данных, полученных в аналитическом разделе, была спроектирована схема алгоритма DBSCAN.

3 Технологическая часть

3.1 Требования к программному обеспечению

Программа должна отвечать следующим требованиям:

— программа получает на вход файл с точками, минимальное количество точек в кластере, минимальное расстояние и количество потоков;

— программа выдает в результате файл с кластеризованными точками;

— алгоритм реализован с помощью параллельных вычислений;

— программа должна измерять реальное время.

3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран *C++* ввиду следующих причин:

— мое желанием расширить свои знания в области применения данного языка;

— возможность измерять реальное время выполнения алгоритма;

— данный язык предоставляет широкие возможности для работы с потоками.

Таким образом, с помощью языка программирования *C++* можно реализовать программное обеспечение, которое соответствует перечисленным выше требованиям.

3.3 Листинг кода

В листинге 3.1 приведена реализация алгоритма DBSCAN. В листинге 3.2 приведена функция кластеризации, используемая в внешнем циклке DBSCAN.

Листинг 3.1 – Реализация алгоритма DBSCAN

```
void DBSCAN::dbscan()
{
    vector<int> tmp;
    prl_calc_mtx(0, m_points.size());
    prl_calc_kernel(0, m_points.size(), tmp);
    for (int &index: tmp)
        erase_vector(unclassified, index);
    tmp.clear();
    prl_calc_border(0, unclassified.size(), tmp);
    for (int &index: tmp)
        erase_vector(unclassified, index);
    tmp.clear();
    prl_calc_noise(0, unclassified.size(), tmp);
    for (int &index: tmp)
        erase_vector(unclassified, index);
    prl_unclustered_init(0, kernel.size());
    while (unclustered.size() != 0)
    {
        vector<int> cluster;
        vector<Check> checker;
        vector<Check> tmp_checker;
        int index = unclustered[0];
        cluster.push_back(index);
        prl_clustering_init(0, m_points.size(), index,
           cluster, checker);
        while (checker.size() != 0)
        {
            prl_clustering_step(0, checker.size(), cluster,
               checker, tmp_checker);
            checker.swap(tmp_checker);
            tmp_checker.clear();
        }
        for (int &index: cluster)
            erase_vector(unclustered, index);
        clusters.push_back(cluster);
    }
}
```

Листинг 3.2 – Функция кластеризации

```
void DBSCAN::prl_clustering_step(
    int start, int end,
    vector<int> &cluster,
    vector<Check> &checker,
    vector<Check> &tmp_checker
)
{
    for (int i = 0; i < checker.size(); i++)
        for (int j = 0; j < m_points.size(); j++)
        {
            bool is_reachable = mtx[checker[i].index][j];
            bool in_cluster = find(
                cluster.begin(), cluster.end(), j
            ) != cluster.end();
            bool is_border = find(
                border.begin(), border.end(), j
            ) != border.end();
            bool is_kernel = find(
                kernel.begin(), kernel.end(), j
            ) != kernel.end();
            if (is_reachable && !in_cluster)
            {
                if (is_border && checker[i].is_kernel)
                    tmp_checker.push_back(Check{j, false});
                else
                    tmp_checker.push_back(Check{j, true});
                cluster.push_back(j);
            }
        }
}
```

3.4 Тестовые данные

В таблице 3.1 приведены тесты для функции, реализующей алгоритм DBSCAN. Тесты выполнялись по методологии черного ящика. Все тесты пройдены успешно.

Таблица 3.1 – Тестовые данные

Точки	**MinPt**	**Eps**	**Количество кластеров**
1 1 1 2 2 1	3	0.5	1
1 1 1 2 2 1	3	1.5	0
1 1 1 2 2 1 5 5 5 6 6 5 10 10	3	1	2

Вывод

На основе схемы, полученной в конструкторском разделе, был реализован и протестирован алгоритм DBSCAN.

4 Исследовательская часть

4.1 Технические характеристики

Тестирование выполнялось на устройстве со следующими техническими характеристиками.

1. Операционная система: Ubuntu Linux 64-bit.
2. Память: 8 GiB.
3. Процессор: AMD Ryzen 5 3550H.

4.2 Время выполнения алгоритма

Время работы функции замерено с помощью ассемблерной инструкции **rdtsc**, которая читает счетчик Time Stamp Counter и возвращает 64-битное количество тиков с момента последнего сброса процессора.

Результаты тестирования приведены в таблице 4.1. На рисунке 4.1 приведена зависимость времени работы алгоритма от количества точек.

Таблица 4.1 – Время работы алгоритма

	Время работы, тики		
Количество точек	**1 поток**	**2 потока**	**4 потока**
10	248703	3426444	3376998
25	2490537	4550885	3937556
50	6295065	7454529	4888677
75	19141605	12871984	8106802
100	36849057	21726349	12530794

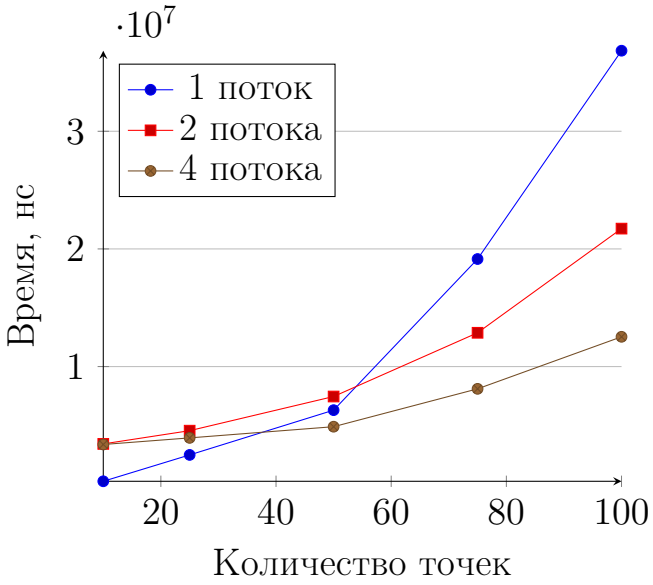

Рисунок 4.1 – Зависимость времени выполнения от количества потоков

Вывод

Согласно полученным при проведении эксперимента данным, при малом количестве входных данных, последовательно работающий алгоритм работает быстрее, чем многопоточные. Это объясняется тем, что на выделение поток тратиться значительное количество ресурсов. С увеличением входных данных, многопоточные начинают обгонять последовательный алгоритм, поскольку время, затраченное на выделение потоков, становиться незначительным, по сравнению со временем, затраченным на вычисления.

ЗАКЛЮЧЕНИЕ

В рамках данной лабораторной работы была достигнута поставленная цель: были изучены понятия параллельных вычислений и с помощью них реализован алгоритм DBSCAN.

Решены все поставленные задачи:

1) изучено понятия параллельных вычислений;
2) реализован алгоритм DBSCAN с использованием параллельных вычислений;
3) проведен сравнительный анализ времени работы алгоритма для различного количества используемых потоков;
4) обоснованы полученные результаты.

Исходя из результатов, полученных в исследовательской части, при большом количестве данных, наболее эффективной является многопоточная реализация алгоритма DBSCAN. При этом, если требуеться обработать небольшое количество данных, лучше использовать последовательную реализацию.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Гергель В. П.* Теория и практика параллельных вычислений. — 2007.
2. *Ершов К.*, *Романова Т.* Анализ и классификация алгоритмов кластеризации // Новые информационные технологии в автоматизированных системах. — 2016. — № 19. — С. 274—279.
3. Потоки [Электронный ресурс]. — Режим доступа: `https://learn.microsoft.com/ru-ru/dotnet/standard/threading/threads-and-threading` (дата обращения 5.11.2022).
4. *Большакова Е. И.*, *Клышинский Э. С.* Автоматическая обработка текстов на естественном языке и компьютерная лингвистика. — 2021.