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 на основе нативных потоков»

Студент ИУ7-51Б _____ Д. В. Шубенина
(Группа) (Подпись, дата) (И. О. Фамилия)

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

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

2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Кластеризация данных является важным инструментом в области машинного обучения. Она позволяет группировать данные на основе их сходства и отделить их от остальных [1].

Целью данной лабораторной работы является получение навыков организации параллельного выполнения операций.

Для достижения поставленной цели необходимо решить следующие задачи:

1) описать нечеткий алгоритм кластеризации c-средних;
2) разработать параллельную версию алгоритма;
3) определить средства программной реализации;
4) реализовать данные алгоритмы;
5) выполнить замеры процессорного времени работы различных реализаций алгоритма и произвести анализ полученных данных.

1 Аналитическая часть

В данном разделе приведена информация о понятии кластеризации и нечетком алгоритме кластеризации с-средних.

1.1 Нечеткий алгоритм кластеризации с-средних

Кластерный анализ — это ряд математических методов интеллектуального анализа данных, предназначенных для разбиения множества исследуемых объектов на компактные группы, называемые кластерами. Под объектами кластерного анализа подразумеваются предметы исследования, нуждающиеся в кластеризации по некоторым признакам. Признаки объектов могут иметь как непрерывные, так и дискретные значения [2].

Метод с-средних — итеративный нечеткий алгоритм кластеризации. В данном методе кластеры являются нечеткими множествами, и каждый объект из выборки исходных данных относится одновременно ко всем кластерам с различной степенью принадлежности. Таким образом, матрица принадлежности объектов к кластерам (или матрица разбиения) содержит не бинарные, а вещественные значения, принадлежащие отрезку $[0; 1]$ [2].

Пусть X — исходный набор данных размера N. Обновление матрицы принадлежности и списка центров кластеров производится в 5 этапов:

1) инициализация матрицы центров кластеров W случайными значениями;

2) инициализация матрицы разбиения $U = (\mu_{ij})$ следующим образом:

$$\mu_{ij}^{(t)} = \frac{1}{\sum\limits_{l=1}^{C} \left(\frac{d_{ij}}{d_{il}}\right)^{\frac{2}{m-1}}}, i = 1, \ldots, N; j, l = 1, \ldots C$$
$$\mu_{ij} = \begin{cases} 1, & \text{если } d_{ij} = 0 \\ 0, & \text{для } l \neq j, \end{cases} \tag{1.1}$$

где t — номер итерации,
C — количество кластеров,
m — показатель нечеткости, регулирующий точность разбиения,
d_{ij} — расстояние от x_i до w_j, $d_{ij} = \|x_i - w_j^{(t)}\|$;

3) увеличить t на 1 и рассчитать матрицу $W^{(t)}$ по формуле (1.2)

$$W_j^{(t)} = \frac{\sum_{i=1}^{N} \left(\mu_{ij}^{(t-1)}\right)^m x_i}{\sum_{i=1}^{N} \left(\mu_{ij}^{(t-1)}\right)^m}, j = 1, \ldots, C; \tag{1.2}$$

4) вычислить матрицу разбиения $U^{(t)}$ согласно соотношению (1.1);

5) если $\|U^{(t)} < U^{(t-1)}\| \geq \varepsilon$ перейти на шаг 3 [3].

1.2 Использование потоков

В данной задаче возможно использование потоков при заполнении матрицы принадлежности: матрица разбивается на n групп строк, где n — количество потоков. Каждая такая группа обрабатывается параллельно. Поскольку элементы матрицы вычисляются независимо друг от друга (см. (1.1)) в использовании средств синхронизации (мьютекс, семафор) нет необходимости.

Вывод

В данном разделе было рассмотрено понятие кластеризации. Также был описан нечеткий алгоритм с-средних.

2 Конструкторская часть

В данном разделе разработаны схемы реализаций нечеткого алгоритма кластеризации c-средних.

2.1 Требования к программному обеспечению

К программному обеспечению предъявлен ряд требований:

1) наличие интерфейса для выбора действий;
2) возможность загрузки массива исходных данных, записанных в текстовый файл;
3) работа с массивами и «нативными» потоками.

2.2 Разработка алгоритмов

На рисунке 2.1 представлена схема последовательного нечеткого алгоритма кластеризации c-средних.

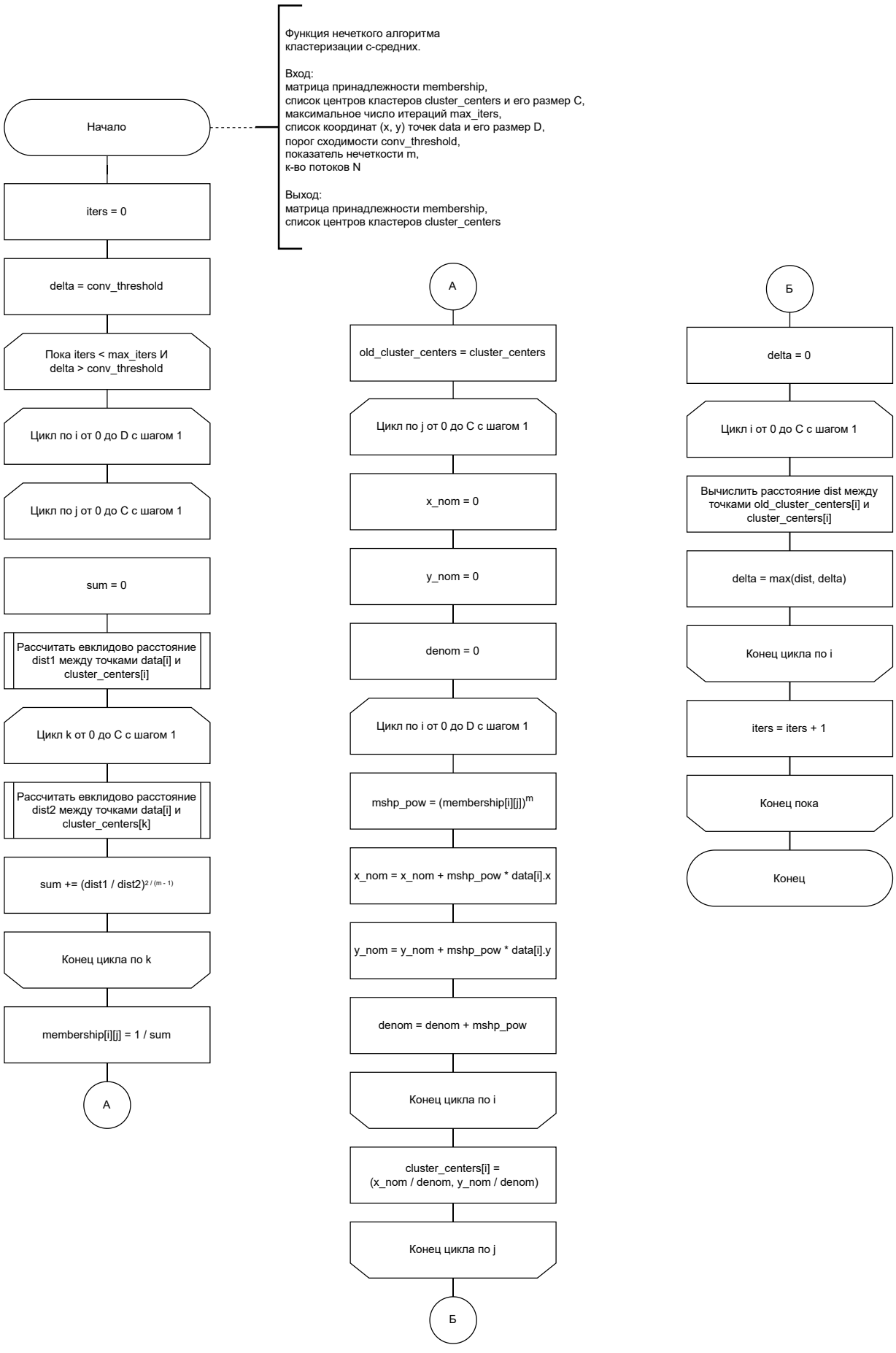

Рисунок 2.1 – Схема последовательного нечеткого алгоритма c-средних

На рисунке 2.2 представлена схема алгоритма главного потока, вызывающего вспомогательные потоки.

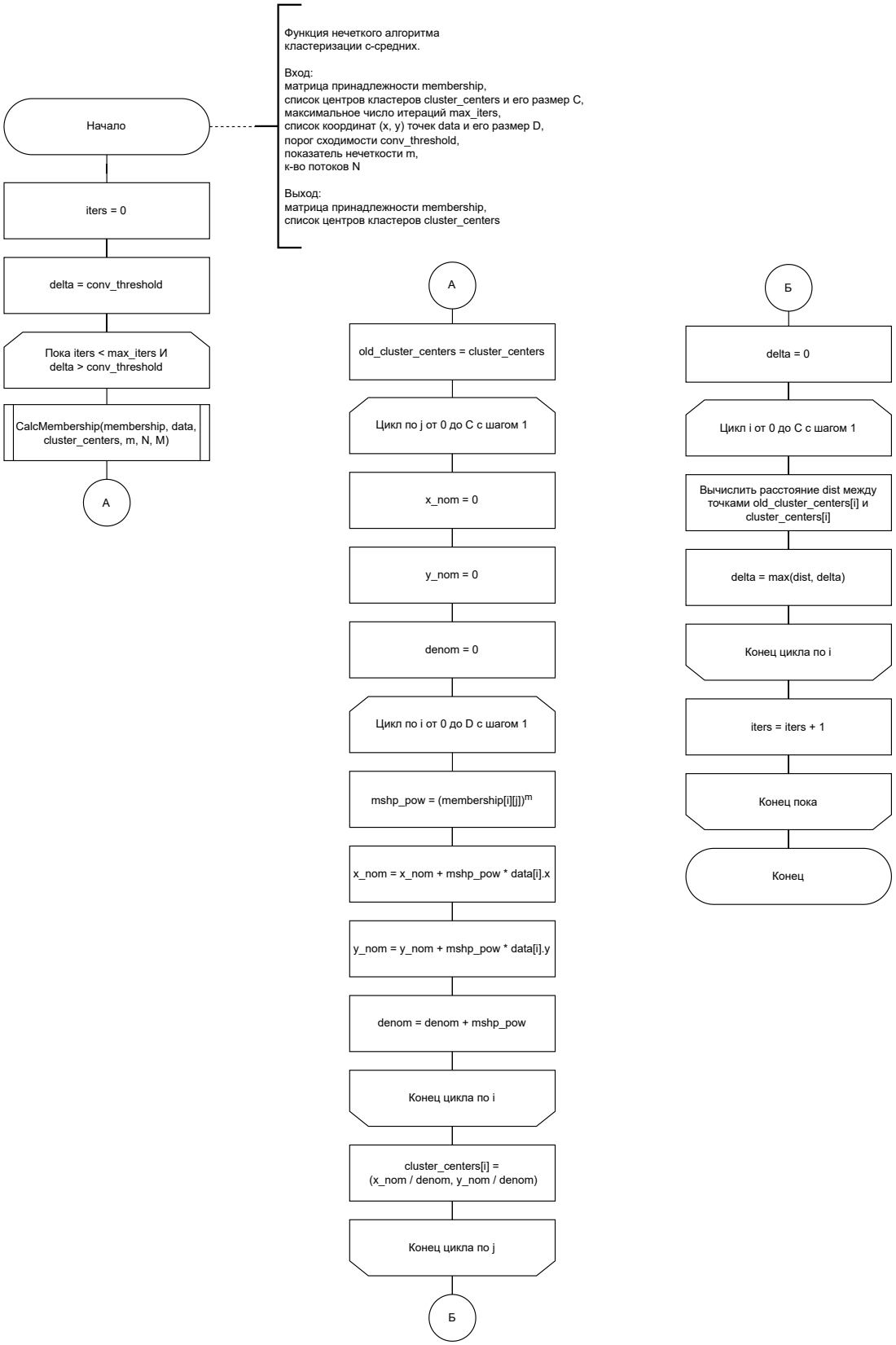

Рисунок 2.2 – Схема алгоритма главного потока, вызывающего вспомогательные потоки

На рисунке 2.3 представлена схема алгоритма, производящего подготовку к вызову вспомогательных потоков и ожидание их завершение.

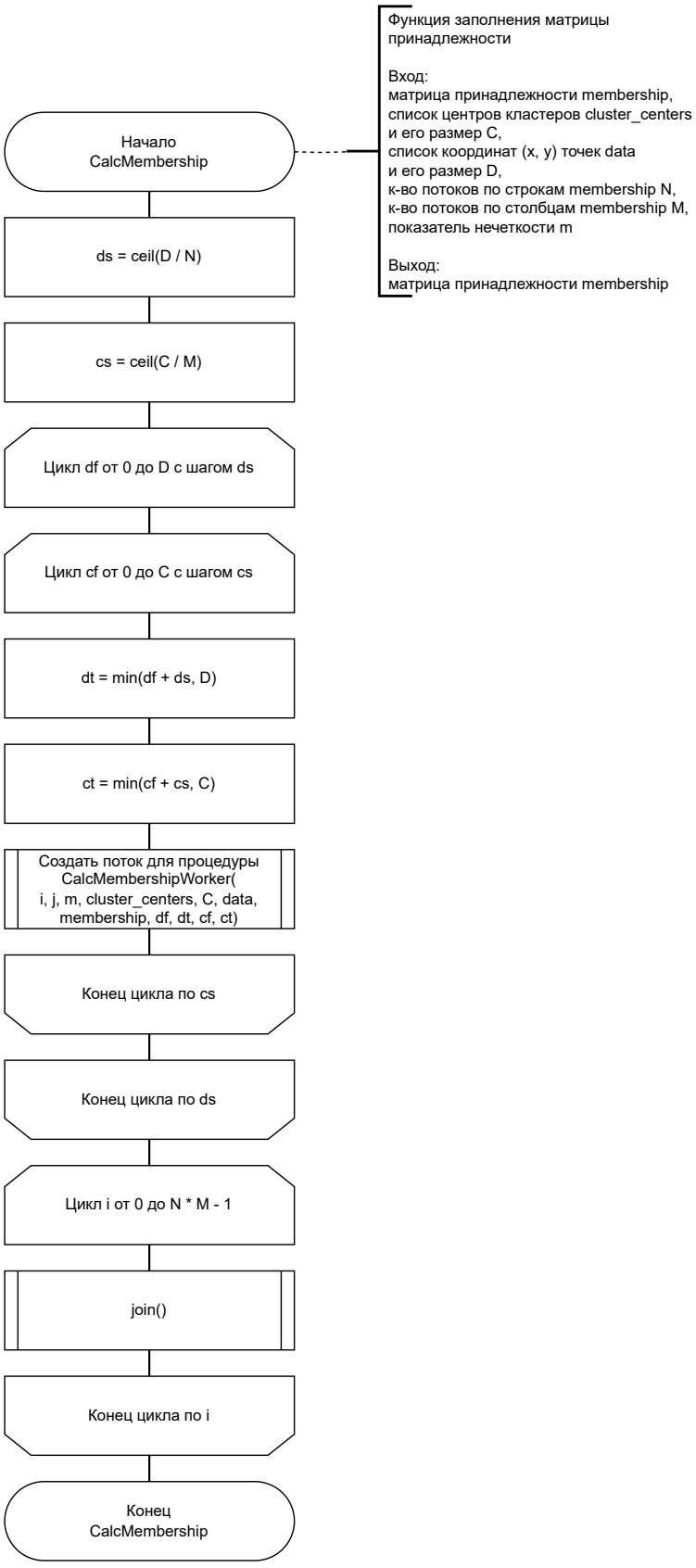

Рисунок 2.3 – Схема алгоритма, производящего подготовку к вызову вспомогательных потоков и ожидание их завершение

На рисунке 2.4 представлена схема алгоритма, выполняющегося внутри вспомогательных потоков для обновления элементов матрицы принадлежности.

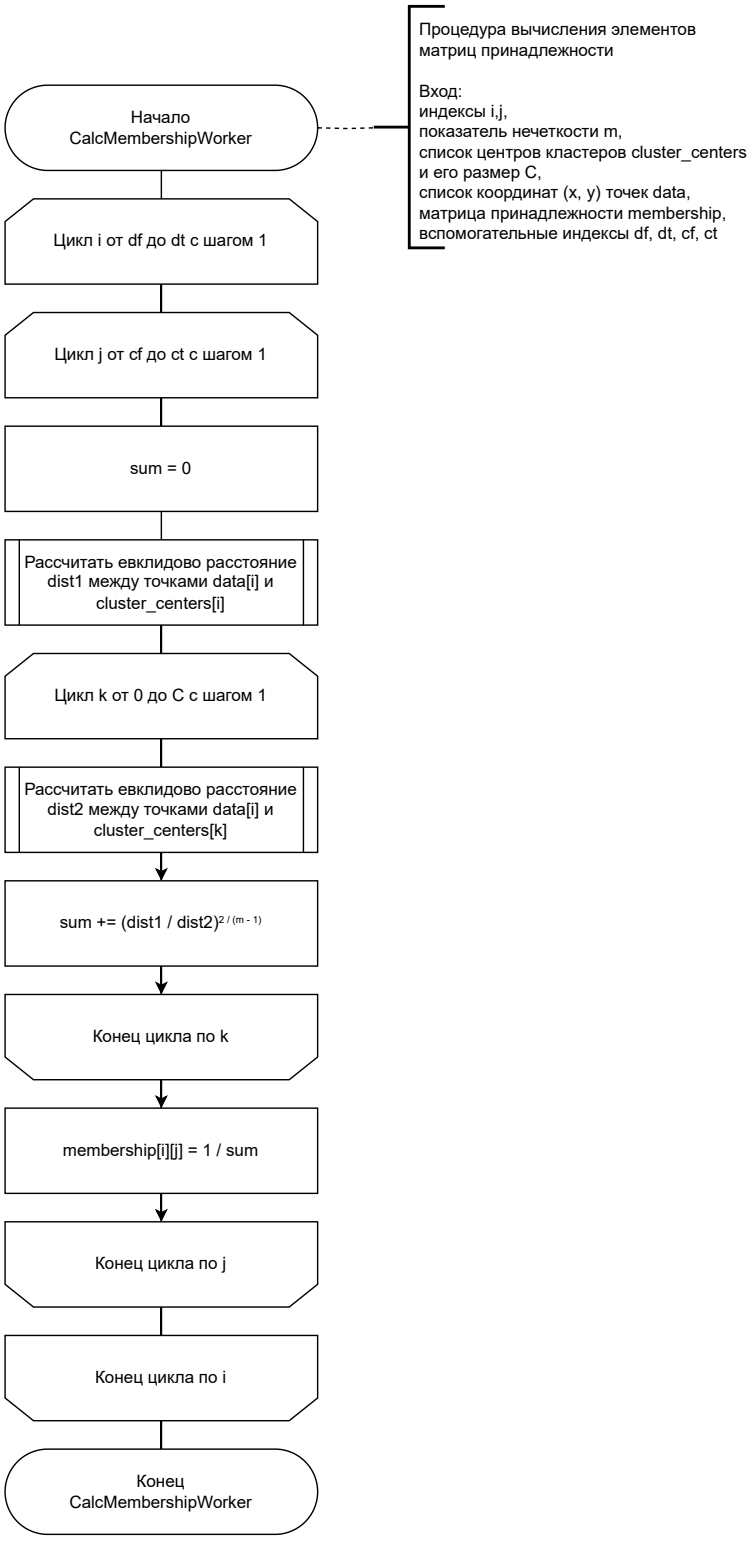

Рисунок 2.4 – Схема алгоритма, выполняющегося внутри вспомогательных потоков для обновления элементов матрицы принадлежности

Вывод

В данном разделе были перечислены требования к программному обеспечению и построены схемы рассматриваемых алгоритмов.

3 Технологическая часть

В данном разделе описаны средства реализации программного обеспечения, а также листинги и функциональные тесты.

3.1 Средства реализации

В качестве языка программирования, используемого при написании данной лабораторной работы, был выбран C++ [4], так как в нем имеется контейнер `std::vector`, представляющий собой динамический массив данных произвольного типа, и библиотека `<ctime>` [5], позволяющая производить замеры процессорного времени.

Для создания потоков и работы с ними был использован класс `thread` из стандартной библиотеки C++. Листинг 3.1 содержит пример работы с описанным классом, каждый экземпляр которого представляет собой поток операционной системы, позволяющий нескольким функциям выполняться одновременно.

Листинг 3.1 – Пример работы с классом thread

```
#include <thread>
#include <iostream>

void func(int &res, int a, int b)
{
    res = a + b;
}

int main()
{
    int res;
    std::thread th(func, std::ref(res), 10, 20);
    th.join();
    std::cout << res << "\n";
    return 0;
}
```

Поток начинает свою работу сразу после создания объекта класса `thread`, запуская функцию, переданную в его конструктор, с переданными туда же параметрами. Данном примере был запущен 1 поток, который выполнит функцию `func`, которая запишет число 30 в переменную `res`.

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

— `main.cpp` — файл, содержащий точку входа в программу;

— `algorithms.cpp` — файл, содержащий последовательную и параллельную реализации нечеткого алгоритма c-средних;

— `measure.cpp` — файл, содержащий функции, замеряющие процессорное время выполнения реализуемых алгоритмов.

3.3 Реализация алгоритмов

На листинге 3.2 представлена реализация последовательной версии нечеткого алгоритма кластеризации c-средних.

Листинг 3.2 – Реализация последовательного алгоритма кластеризации c-средних

```
1  void c_means(
2          membership_t &membership, point_vec_t &cluster_centers,
3          const point_vec_t &data,
4          double m, double conv_threshold, int max_iters)
5  {
6      int iters = 0;
7      double delta = conv_threshold + 1.0;
8      while (iters < max_iters && delta > conv_threshold)
9      {
10         for (size_t i = 0; i < data.size(); ++i)
11         {
12             for (size_t j = 0; j < cluster_centers.size(); ++j)
13             {
14                 double sum = 0.0;
15                 double dist1 = sqrt(pow(data[i][0] -
                       cluster_centers[j][0], 2) +
16                                     pow(data[i][1] -
                                           cluster_centers[j][1],
                                           2));
17                 for (size_t k = 0; k < cluster_centers.size();
                       ++k)
18                 {
```

```
19                      double dist2 = sqrt(pow(data[i][0] -
                           cluster_centers[k][0], 2) +
20                                          pow(data[i][1] -
                                               cluster_centers[k][1],
                                               2));
21                      sum += pow(dist1 / dist2, 2.0 / (m - 1.0));
22                  }
23                  membership[i][j] = 1.0 / sum;
24              }
25          }
26          auto old_cluster_centers = cluster_centers;
27          for (size_t j = 0; j < cluster_centers.size(); ++j)
28          {
29              double x_nom = 0.0, y_nom = 0.0, denom = 0.0;
30              for (size_t i = 0; i < data.size(); ++i)
31              {
32                  double membership_pow_m = pow(membership[i][j],
                        m);
33                  x_nom += membership_pow_m * data[i][0];
34                  y_nom += membership_pow_m * data[i][1];
35                  denom += membership_pow_m;
36              }
37              cluster_centers[j] = {x_nom / denom, y_nom / denom};
38          }
39          delta = 0.0;
40          for (size_t i = 0; i < cluster_centers.size(); ++i)
41          {
42              double distance = sqrt(pow(old_cluster_centers[i][0]
                    - cluster_centers[i][0], 2) +
43                                     pow(old_cluster_centers[i][1]
                                        - cluster_centers[i][1],
                                        2));
44              if (distance > delta)
45                  delta = distance;
46          }
47          ++iters;
48      }
49  }
```

На листинге 3.2 представлена реализация алгоритма основного потока, запускающего вспомогательного потоки.

Листинг 3.3 – Функция основного потока, запускающего вспомогательные потоки

```
void c_means_parallel(
        membership_t &membership, point_vec_t &cluster_centers,
        const point_vec_t &data,
        double m, double conv_threshold, int max_iters,
        int n_threads)
{
    int iters = 0;
    double delta = conv_threshold + 1.0;
    while (iters < max_iters && delta > conv_threshold)
    {
        calc_membership(membership, data, cluster_centers, m,
           n_threads);

        auto old_cluster_centers = cluster_centers;
        for (size_t j = 0; j < cluster_centers.size(); ++j)
        {
            double x_nom = 0.0, y_nom = 0.0, denom = 0.0;
            for (size_t i = 0; i < data.size(); ++i)
            {
                double membership_pow_m = pow(membership[i][j],
                   m);
                x_nom += membership_pow_m * data[i][0];
                y_nom += membership_pow_m * data[i][1];
                denom += membership_pow_m;
            }
            cluster_centers[j] = {x_nom / denom, y_nom / denom};
        }
        delta = 0.0;
        for (size_t i = 0; i < cluster_centers.size(); ++i)
        {
            double distance = sqrt(pow(old_cluster_centers[i][0]
               - cluster_centers[i][0], 2) +
                                   pow(old_cluster_centers[i][1]
                                      - cluster_centers[i][1],
                                      2));
            if (distance > delta)
                delta = distance;
        }
        ++iters;
```

```
35      }
36  }
```

На листинге 3.4 представлена реализация алгоритма функции, вызывающей вспомогательные потоки вычисления значений элементов матрицы принадлежности. Для продолжения работы с матрицей принадлежности, содержащей обновленные значения, необходимо дождаться завершения работы всех потоков, выполняющих обновление элементов матрицы.

Листинг 3.4 – Функция, вызывающая вспомогательные потоки вычисления значений элементов матрицы принадлежности

```
1  static void calc_membership(
2      membership_t &membership,
3      const point_vec_t &data, point_vec_t &cluster_centers,
           double m,
4      int n_threads)
5  {
6      std::vector<std::thread> threads;
7
8      int data_step = ceil(float(data.size()) / float(n_threads));
9      for (size_t data_from = 0; data_from < data.size();
           data_from += data_step)
10     {
11         int data_to = std::min<int>(data_from + data_step,
               data.size());
12         threads.emplace_back(
13             calc_membership_worker,
14             std::ref(data),
15             std::ref(cluster_centers),
16             std::ref(membership),
17             data_from, data_to, m);
18     }
19     for (auto &thr : threads)
20     {
21         if (thr.joinable())
22             thr.join();
23     }
24 }
```

На листинге 3.5 представлена реализация алгоритма вычисления значений элементов матрицы принадлежности во вспомогательном потоке.

Листинг 3.5 – Реализация алгоритма вычисления значений элементов матрицы принадлежности во вспомогательном потоке

```
static void calc_membership_worker(
    const point_vec_t &data,
    const point_vec_t &cluster_centers,
    membership_t &membership,
    int data_from, int data_to, double m)
{
    for (int i = data_from; i < data_to; ++i)
    {
        for (size_t j = 0; j < cluster_centers.size(); ++j)
        {
            double sum = 0.0;
            double dist1 = sqrt(
                pow(data[i][0] - cluster_centers[j][0], 2) +
                pow(data[i][1] - cluster_centers[j][1], 2));
            for (size_t k = 0; k < cluster_centers.size(); ++k)
            {
                double dist2 = sqrt(
                    pow(data[i][0] - cluster_centers[k][0], 2) +
                    pow(data[i][1] - cluster_centers[k][1], 2));
                sum += pow(dist1 / dist2, 2.0 / (m - 1.0));
            }
            membership[i][j] = 1.0 / sum;
        }
    }
}
```

3.4 Функциональные тесты

В таблице 3.1 представлены результаты функционального тестирования реализованных алгоритмов кластеризации для двух наборов точек:

1) набор точек 1: $[(0, 0), (0, 2)]$,

2) набор точек 2: $[(0, 0), (0, 2), (10, 40)]$.

Многопоточная версия алгоритма тестировалась при числе потоков 10. Также некоторые параметры алгоритмов были заданы заранее:

— показатель нечеткости `m = 2`,

— максимальное к-во итераций `max_iters = 100`,

— порог сходимости `conv_threshold = 1`.

Все тесты пройдены успешно.

Таблица 3.1 – Результаты функционального тестирования

Входные данные		Результат	
Точки	К-во класт.	Послед.	Парал.
Набор точек 1	1	$\begin{pmatrix} 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \end{pmatrix}$
Набор точек 1	2	$\begin{pmatrix} 0.296 & 0.704 \\ 0.699 & 0.301 \end{pmatrix}$	$\begin{pmatrix} 0.296 & 0.704 \\ 0.699 & 0.301 \end{pmatrix}$
Набор точек 2	3	$\begin{pmatrix} 0.999 & 0.001 & 0.001 \\ 0.999 & 0.001 & 0.001 \\ 0.000 & 0.020 & 0.980 \end{pmatrix}$	$\begin{pmatrix} 0.999 & 0.001 & 0.001 \\ 0.999 & 0.001 & 0.001 \\ 0.000 & 0.020 & 0.980 \end{pmatrix}$

Вывод

В данном разделе были рассмотрены средства реализации, а также представлен листинг реализаций последовательного и параллельного нечеткого алгоритма кластеризации с-средних.

4 Исследовательская часть

В данном разделе приведены технические характеристики устройства, на котором проводилось измерение времени работы программного обеспечения, а также результаты замеров времени.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени:

— процессор: AMD Ryzen 7 5800X @ 3.800 ГГц, 8 физ. ядер, 16 лог. ядер;

— оперативная память: 32 ГБайт.

— операционная система: Manjaro Linux x86_64 (версия ядра Linux 6.5.12-1-MANJARO).

Измерения проводились на стационарном компьютере. Во время проведения измерений устройство было нагружено только системными приложениями.

4.2 Демонстрация работы программы

На рисунке 4.1 продемонстрирована работа программы для случая, когда пользователь выбрал пункт 1 «Кластеризация методом с-средних» и ввел следующие значения:

— число кластеров — 2,

— значение показателя нечеткости — 3,

— значение порога сходимости — 0.3,

— максимально к-во итераций — 10,

— число дополнительных потоков — 16.

```
                Меню
1. Кластеризация методом c-средних:
    а) однопоточная;
    б) многопоточная.
2. Редактировать файл, содержащий множество точек.
3. Произвести замеры по времени реализуемых алгоритмов.
0. Выход.

Выберите опцию (0-3): 1

Введите число кластеров (1-3): 2
Введите значение показателя нечеткости: 3
Введите значение порога сходимости: 0.1
Введите максимальное к-во итераций: 10
Введите число дополнительных потоков (для многопоточной версии): 16
Матрица принадлежности
 0.976  0.024
 0.975  0.025
 0.000  1.000
Центроиды кластеров: [ [ 0.000 0.999 ] [ 10.000 39.999 ] ]
Матрица принадлежности
 0.976  0.024
 0.975  0.025
 0.000  1.000
Центроиды кластеров: [ [ 0.000 0.999 ] [ 10.000 39.999 ] ]

                Меню
1. Кластеризация методом c-средних:
    а) однопоточная;
    б) многопоточная.
2. Редактировать файл, содержащий множество точек.
3. Произвести замеры по времени реализуемых алгоритмов.
0. Выход.

Выберите опцию (0-3): 0
```

Рисунок 4.1 – Демонстрация работы программы

4.3 Временные характеристики

Исследование временных характеристик реализуемых алгоритмов производилось 2 раза:

1) при изменении числа потоков $1, 2, 4, \ldots, 128$ для набора данных из 10000 точек;

2) при изменении размера набора данных от 5000 до 30000 с шагом 5000 и

при использовании 1 вспомогательного потока.

Наборы данных генерировались из равномерного распределения.

На рисунке 4.2 представлены результаты измерения времени работы реализуемых алгоритмов при варьировании числа потоков; таблица 4.1 содержит значения, по которой был построен данный график.

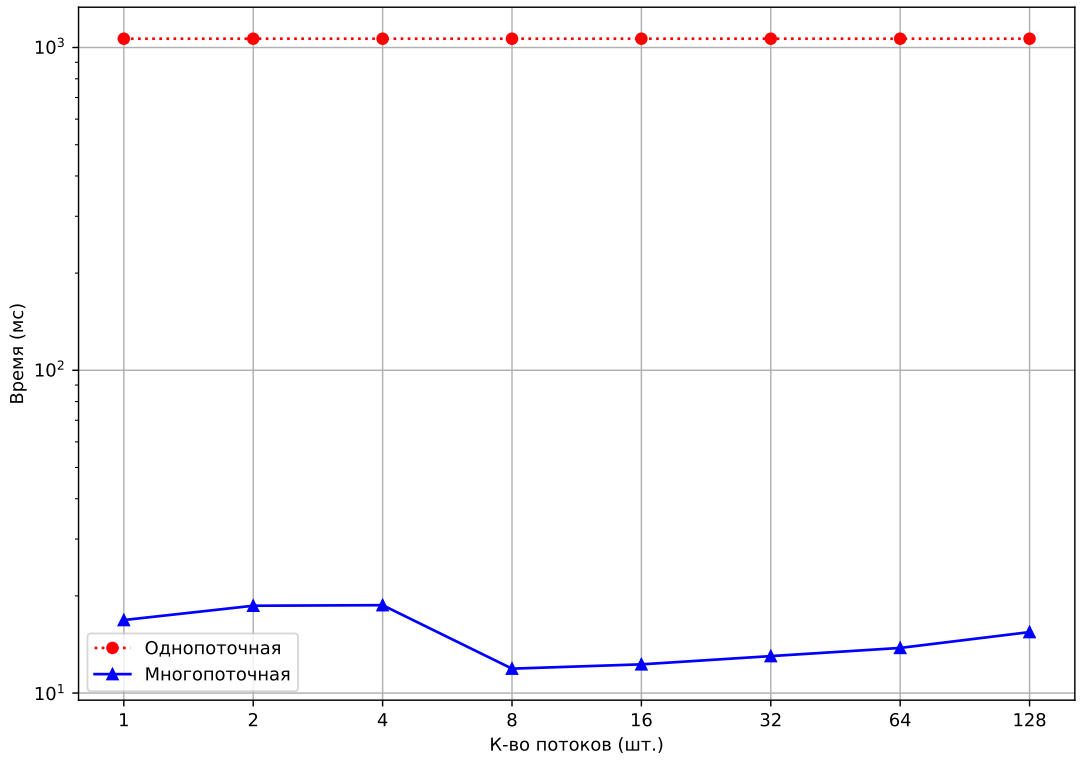

Рисунок 4.2 – Сравнение последовательной и параллельной реализаций нечеткого алгоритма кластеризации с-средних при изменении числа потоков

Таблица 4.1 – Результаты измерения времени работы реализуемых алгоритмов при варьировании числа потоков

К-во потоков (шт.)	Размер (элем.)	Время (мс)	
		Послед.	Парал.
2	10000	1 064 832.83	18 643.05
4	10000	1 064 832.83	18 717.83
8	10000	1 064 832.83	11 900.51
16	10000	1 064 832.83	12 271.80
32	10000	1 064 832.83	13 015.43
64	10000	1 064 832.83	13 797.37
128	10000	1 064 832.83	15 447.34

На рисунке 4.3 показаны результаты измерения времени работы параллельного нечеткого алгоритма кластеризации c-средних в зависимости от числа потоков.

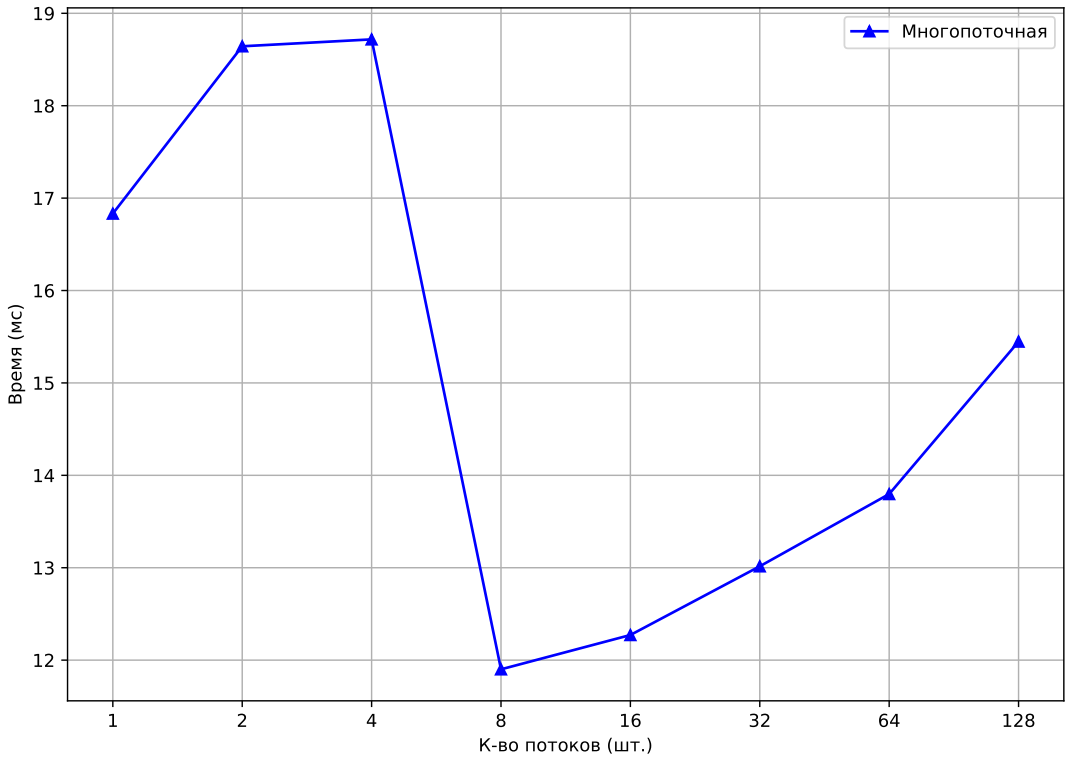

Рисунок 4.3 – График времени работы параллельного нечеткого алгоритма кластеризации c-средних

На рисунке 4.4 представлены результаты измерения времени работы последовательной и параллельной версий рассматриваемого алгоритма в зависимости от размера набора данных; в таблице 4.2 приведены значения, по которым строился данный график.

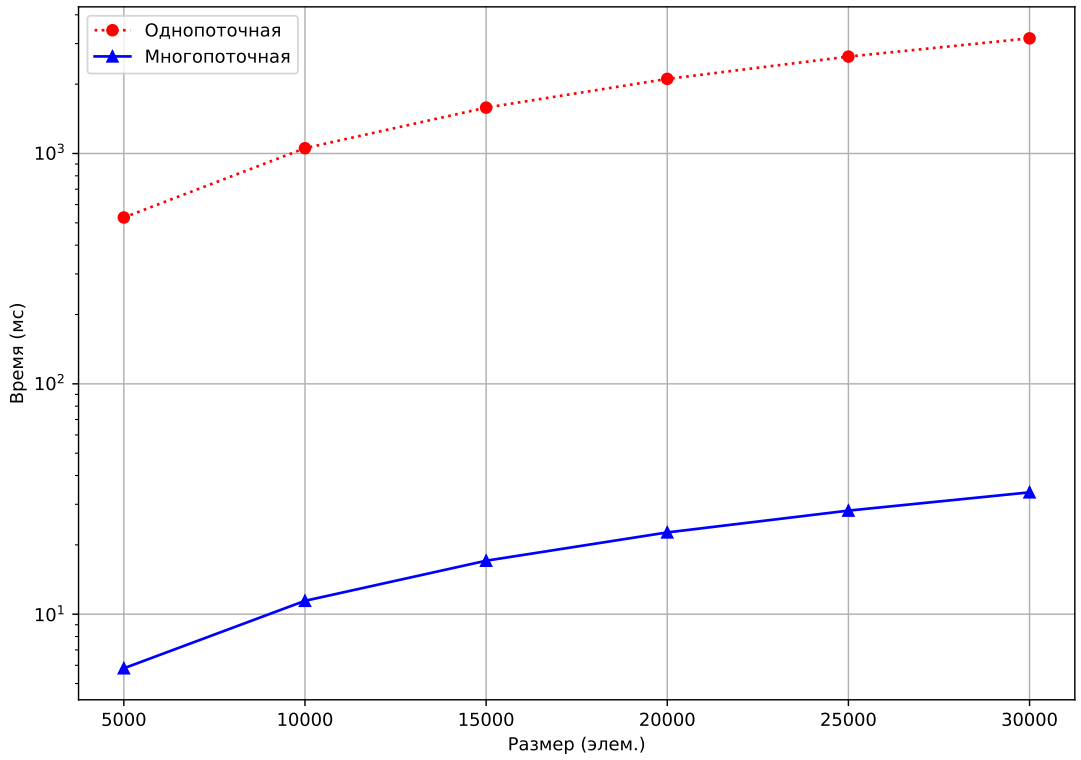

Рисунок 4.4 – Сравнение последовательной и параллельной реализаций нечеткого алгоритма кластеризации с-средних при изменении размера датасета

Таблица 4.2 – Результаты измерения времени работы реализуемых алгоритмов при варьировании размера датасета

К-во потоков (шт.)	Размер (элем.)	Время (мс)	
		Послед.	Парал.
1	10000	1 054 345.96	11 429.15
1	15000	1 581 842.77	17 065.27
1	20000	2 107 652.32	22 645.99
1	25000	2 634 434.62	28 160.84
1	30000	3 161 812.22	33 805.39

4.4 Вывод

В результате исследования реализуемых алгоритмов по времени выполнения можно сделать следующие выводы:

1) при варьировании размера датасета последовательная версия нечеткого алгоритма кластеризации c-средних выполнялась в среднем в 92.6 раз дольше, чем параллельная;

2) при варьировании числа потоков последовательная версия нечеткого алгоритма кластеризации c-средних выполнялась в среднем 72.7 раз дольше, чем параллельная;

3) наименьшее время работы многопоточной реализации алгоритма достигается при 8 вспомогательных потоках; наибольшее же время выполнения алгоритма достигается при 4 потоках.

Таким образом, рекомендуется использование 8 вспомогательных потоков, т. к. при таком количестве временные затраты на создание потоков, переключение аппаратного контекста и синхронизацию ниже, чем получаемая скорость обработки набора данных.

ЗАКЛЮЧЕНИЕ

В результате выполнения лаботарторной работы по исследованию алгоритмов сортировок решены следующие задачи:

1) описан нечеткий алгоритм кластеризации с-средних;
2) разработана параллельная версия алгоритма;
3) определены средства программной реализации;
4) реализованы последовательная и параллельная версия алгоритма;
5) проведен сравнительный анализ процессорного времени выполнения реализованных алгоритмов:

—

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Каримов К. Х.*, *Василий Е. А.* Теоретические основы кластеризации данных // Актуальные вопросы фундаментальных и прикладных научных исследований : Сборник научных статей по материалам II Международной научно-практической конференции, Уфа, 19 мая 2023 года. Том Часть 2. — Уфа : Общество с ограниченной ответственностью “Научно-издательский центр «Вестник науки»”, 2023. — С. 242—247. — EDN XNJGQS.

2. *Лосев Д. Г.* Разработка и сравнение параллельных реализаций итеративных алгоритмов кластеризации // Наука молодых - будущее России : сборник научных статей 6-й Международной научной конференции перспективных разработок молодых ученых (9-10 декабря 2021 года), в 5 томах. Т. 4. — Курск : Юго-Зап. гос. ун-т, 2021. — С. 71—74.

3. *Hung M.-C.*, *Yang D.-L.* An efficient Fuzzy C-Means clustering algorithm // Proceedings 2001 IEEE International Conference on Data Mining. — 2001. — С. 225—232. — DOI: 10.1109/ICDM.2001.989523.

4. C++ language. — [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/language (дата обращения: 12.12.2023).

5. Standard library header <ctime>. — [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/header/ctime (дата обращения: 17.12.2023).