



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  
(Группа)

\_\_\_\_\_  
(Подпись, дата)

Д. В. Шубенина  
(И. О. Фамилия)

Преподаватель

\_\_\_\_\_  
(Подпись, дата)

Л. Л. Волкова  
(И. О. Фамилия)

Преподаватель

\_\_\_\_\_  
(Подпись, дата)

Ю. В. Строганов  
(И. О. Фамилия)

2023 г.

# СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ</b>	<b>3</b>
<b>1 Аналитическая часть</b>	<b>4</b>
1.1 Нечеткий алгоритм кластеризации с-средних . . . . .	4
1.2 Использование потоков . . . . .	5
<b>2 Конструкторская часть</b>	<b>6</b>
2.1 Требования к программному обеспечению . . . . .	6
2.2 Разработка алгоритмов . . . . .	6
<b>3 Технологическая часть</b>	<b>12</b>
3.1 Средства реализации . . . . .	12
3.2 Сведения о модулях программы . . . . .	13
3.3 Реализация алгоритмов . . . . .	13
3.4 Функциональные тесты . . . . .	17
<b>4 Исследовательская часть</b>	<b>19</b>
4.1 Технические характеристики . . . . .	19
4.2 Демонстрация работы программы . . . . .	19
4.3 Временные характеристики . . . . .	20
4.4 Вывод . . . . .	20
<b>ЗАКЛЮЧЕНИЕ</b>	<b>21</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ</b>	<b>22</b>

## 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 с-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:

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

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)

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

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 столбцов, где  $n$  — количество потоков. Каждая такая группа обрабатывается параллельно. Поскольку элементы матрицы вычисляются независимо друг от друга (см. формулу (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 с-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 с-средних.

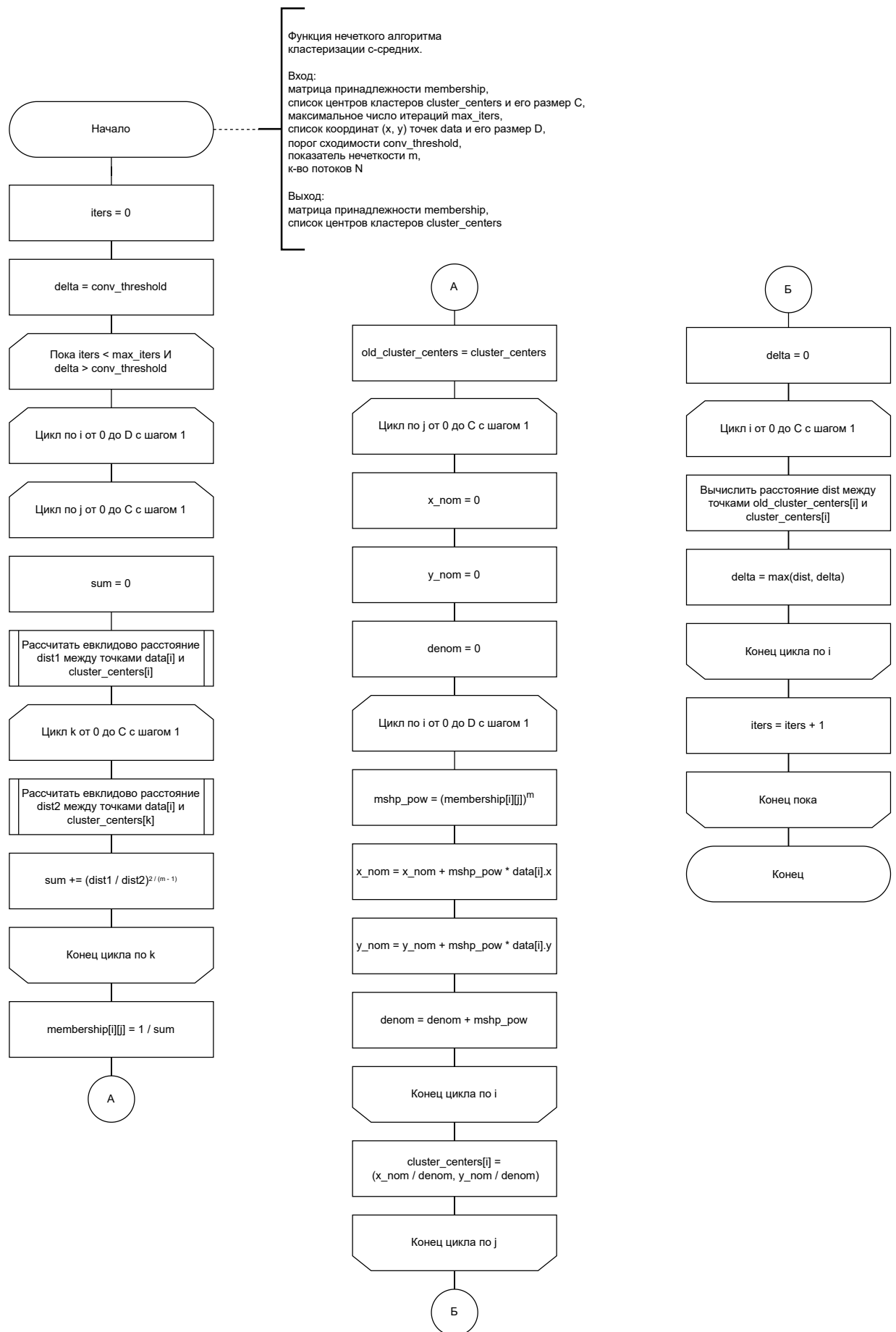


Рисунок 2.1 – Схема последовательного нечеткого алгоритма с-средних

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

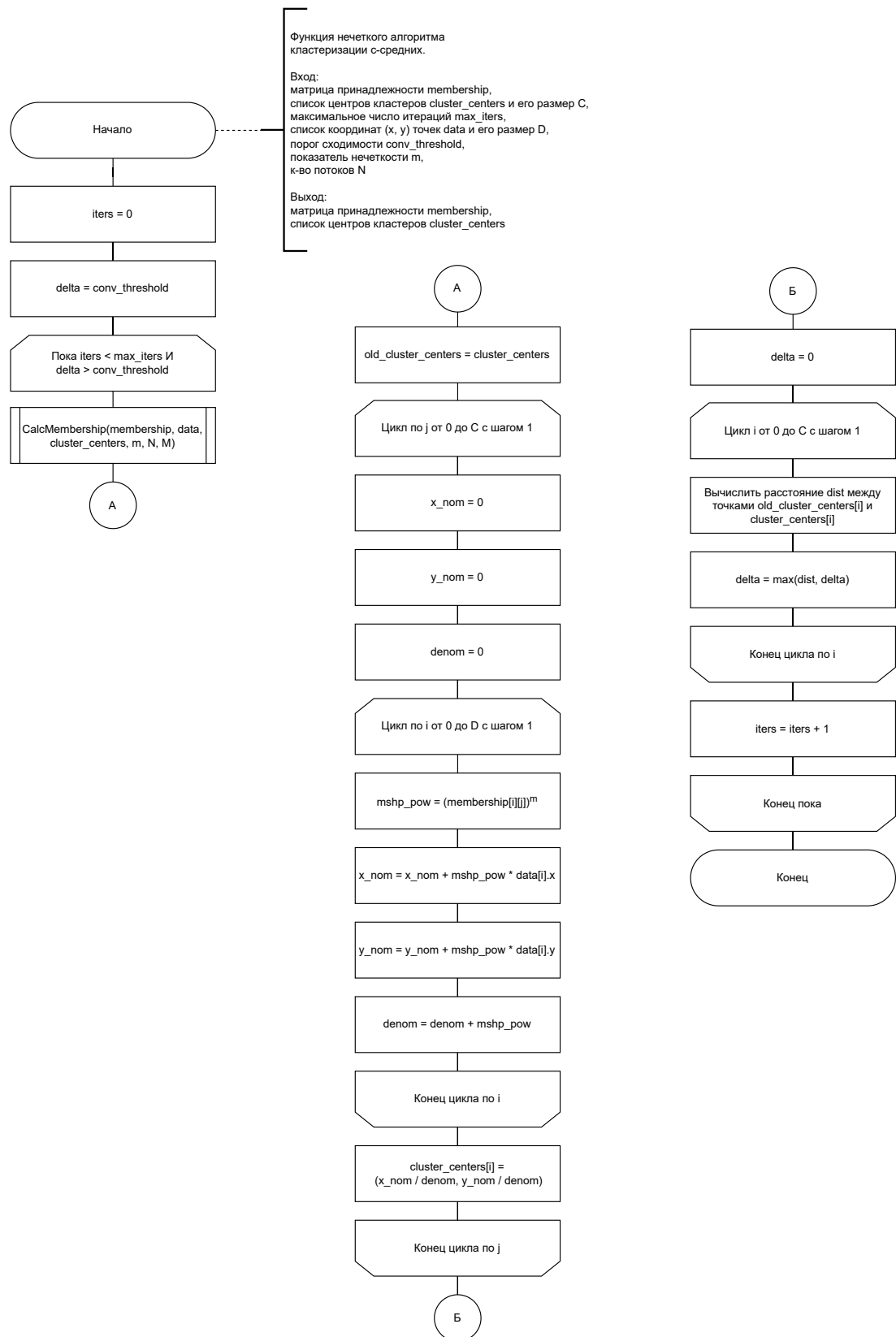


Рисунок 2.2 – Схема алгоритма главного потока, вызывающего вспомогательные потоки



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

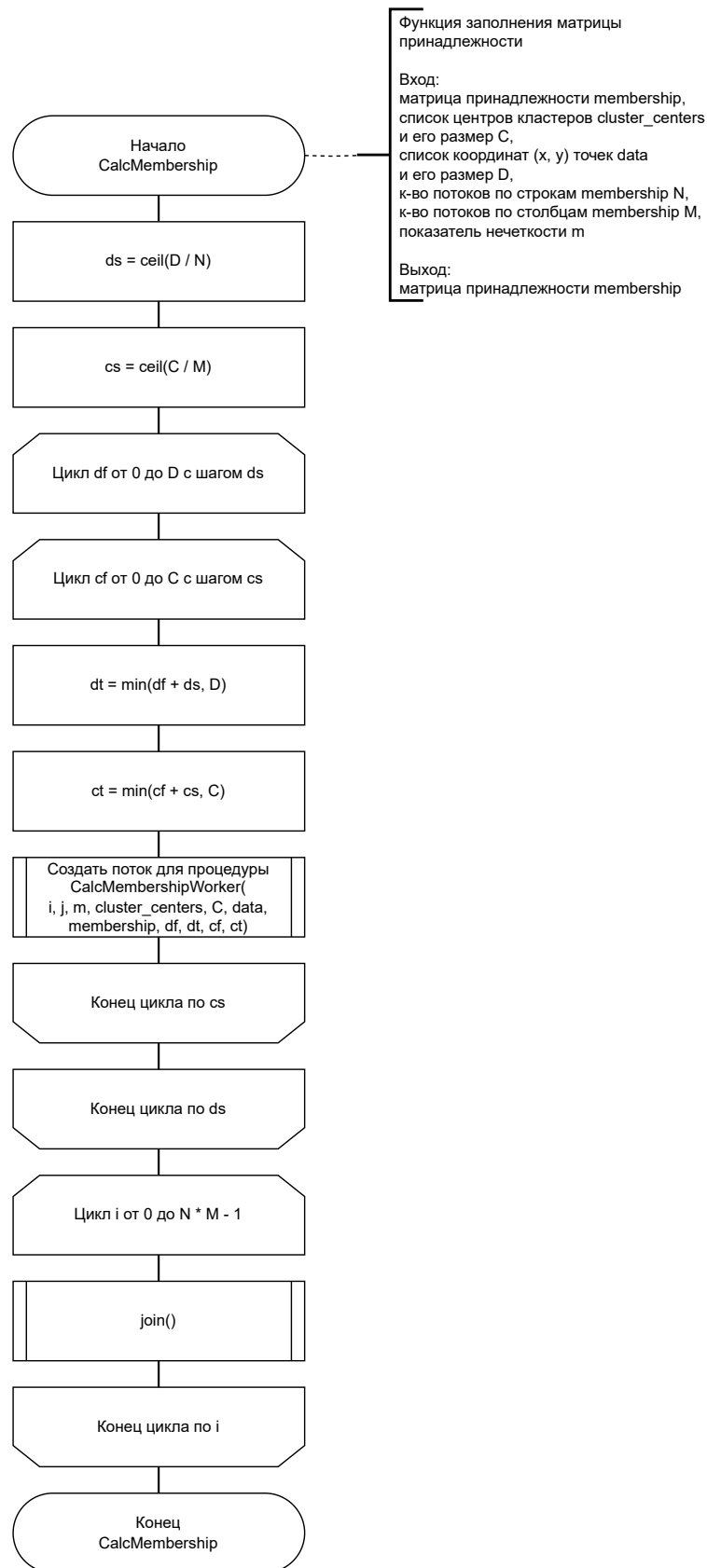


Рисунок 2.3 – Схема алгоритма, производящего подготовку к вызову вспомогательных потоков и ожидание их завершение

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

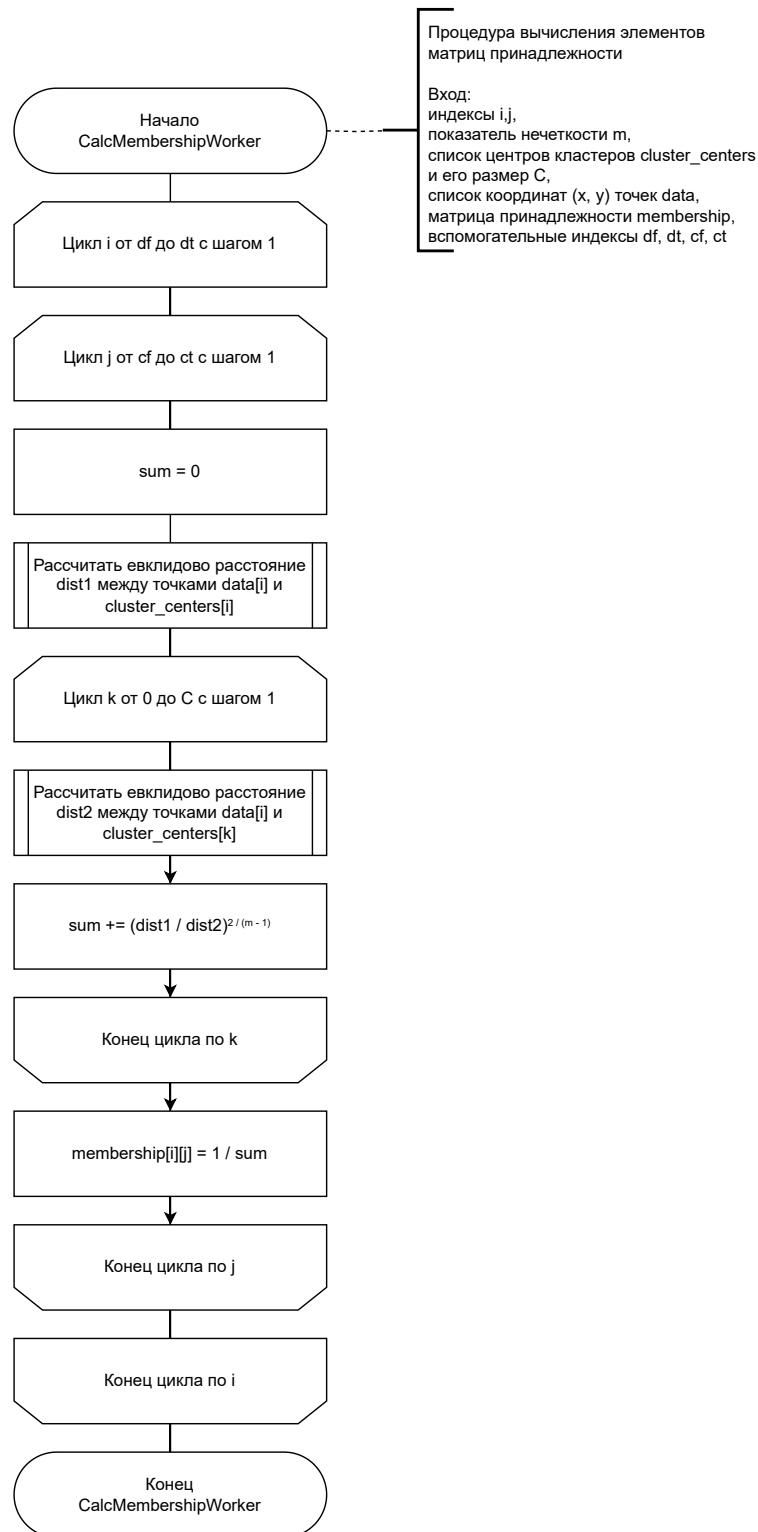


Рисунок 2.4 – Схема алгоритма, выполняющегося внутри вспомогательных потоков для обновления элементов матрицы принадлежности

## 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`

```
1  #include <thread>
2  #include <iostream>
3
4  void func(int &res, int a, int b)
5  {
6      res = a + b;
7  }
8
9  int main()
10 {
11     int res;
12     std::thread th(func, std::ref(res), 10, 20);
13     th.join();
14     std::cout << res << "\n";
15     return 0;
16 }
```

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 с-средних;
- `measure.cpp` — файл, содержащий функции, измеряющие процессорное время выполнения реализуемых алгоритмов.

## 3.3 Реализация алгоритмов

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

Листинг 3.2 – Реализация последовательного алгоритма кластеризации с-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] -  
16                     cluster_centers[j][0], 2) +  
17                     pow(data[i][1] -  
18                     cluster_centers[j][1],  
19                     2));  
20                 for (size_t k = 0; k < cluster_centers.size();  
21                     ++k)  
22                 {
```

```

19         double dist2 = sqrt(pow(data[i][0] -
20                                cluster_centers[k][0], 2) +
21                                pow(data[i][1] -
22                                    cluster_centers[k][1],
23                                    2));
24         sum += pow(dist1 / dist2, 2.0 / (m - 1.0));
25     }
26     membership[i][j] = 1.0 / sum;
27 }
28
29 auto old_cluster_centers = cluster_centers;
30 for (size_t j = 0; j < cluster_centers.size(); ++j)
31 {
32     double x_nom = 0.0, y_nom = 0.0, denom = 0.0;
33     for (size_t i = 0; i < data.size(); ++i)
34     {
35         double membership_pow_m = pow(membership[i][j],
36                                         m);
37         x_nom += membership_pow_m * data[i][0];
38         y_nom += membership_pow_m * data[i][1];
39         denom += membership_pow_m;
40     }
41     cluster_centers[j] = {x_nom / denom, y_nom / denom};
42 }
43
44 delta = 0.0;
45 for (size_t i = 0; i < cluster_centers.size(); ++i)
46 {
47     double distance = sqrt(pow(old_cluster_centers[i][0]
48                                - cluster_centers[i][0], 2) +
49                                pow(old_cluster_centers[i][1]
50                                    - cluster_centers[i][1],
51                                    2));
52
53     if (distance > delta)
54         delta = distance;
55 }
56 ++iters;
57 }
58 }

```

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

Листинг 3.3 – Функция основного потока, запускающего вспомогательные потоки

```
1 void c_means_parallel(  
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     int n_threads)  
6 {  
7     int iters = 0;  
8     double delta = conv_threshold + 1.0;  
9     while (iters < max_iters && delta > conv_threshold)  
10    {  
11        std::vector<std::thread> threads;  
12        calc_membership(membership, data, cluster_centers, m,  
13            n_threads);  
14  
15        auto old_cluster_centers = cluster_centers;  
16        for (size_t j = 0; j < cluster_centers.size(); ++j)  
17        {  
18            double x_nom = 0.0, y_nom = 0.0, denom = 0.0;  
19            for (size_t i = 0; i < data.size(); ++i)  
20            {  
21                double membership_pow_m = pow(membership[i][j],  
22                    m);  
23                x_nom += membership_pow_m * data[i][0];  
24                y_nom += membership_pow_m * data[i][1];  
25                denom += membership_pow_m;  
26            }  
27            cluster_centers[j] = {x_nom / denom, y_nom / denom};  
28        }  
29        delta = 0.0;  
30        for (size_t i = 0; i < cluster_centers.size(); ++i)  
31        {  
32            double distance = sqrt(pow(old_cluster_centers[i][0]  
33                - cluster_centers[i][0], 2) +  
34                pow(old_cluster_centers[i][1]  
35                    - cluster_centers[i][1],  
36                    2));  
37            if (distance > delta)  
38                delta = distance;  
39        }  
40    }  
41 }
```

```

35         ++iters;
36     }
37 }

```

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,
4      double m,
5      int n_threads)
6  {
7
8      std::vector<std::thread> threads;
9
10     int data_step = ceil(float(data.size()) / float(n_threads));
11     for (size_t data_from = 0; data_from < data.size();
12         data_from += data_step)
13     {
14         int data_to = std::min<int>(data_from + data_step,
15                                     data.size());
16         threads.emplace_back(
17             calc_membership_worker,
18             std::ref(data),
19             std::ref(cluster_centers),
20             std::ref(membership),
21             data_from, data_to, m);
22     }
23     for (auto &thr : threads)
24         thr.join();
25 }

```

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



```

1 static void calc_membership_worker(
2     const point_vec_t &data,
3     const point_vec_t &cluster_centers,
4     membership_t &membership,
5     int data_from, int data_to, double m)
6 {
7     for (int i = data_from; i < data_to; ++i)
8     {
9         for (size_t j = 0; j < cluster_centers.size(); ++j)
10        {
11            double sum = 0.0;
12            double dist1 = sqrt(
13                pow(data[i][0] - cluster_centers[j][0], 2) +
14                pow(data[i][1] - cluster_centers[j][1], 2));
15            for (size_t k = 0; k < cluster_centers.size(); ++k)
16            {
17                double dist2 = sqrt(
18                    pow(data[i][0] - cluster_centers[k][0], 2) +
19                    pow(data[i][1] - cluster_centers[k][1], 2));
20                sum += pow(dist1 / dist2, 2.0 / (m - 1.0));
21            }
22            membership[i][j] = 1.0 / sum;
23        }
24    }
25 }

```

### 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 с-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 с-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

## 4.4 Вывод

## ЗАКЛЮЧЕНИЕ

В результате выполнения лабораторной 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).