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»

Отчёт

по лабораторной работе №4

Название: Нечёткий алгоритм k-means

Дисциплина: Анализ алгоритмов

Студент ИУ7-52Б (Группа) _____ (Подпись, дата) Н. В. Ляпина (И.О. Фамилия)

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

Москва, 2022

Содержание

Введение

Многопоточность — способность центрального процессора (CPU) или одного ядра в многоядерном процессоре одновременно выполнять несколько процессов или потоков, соответствующим образом поддерживаемых операционной системой.

Этот подход отличается от многопроцессорности, так как многопоточность процессов и потоков совместно использует ресурсы одного или нескольких ядер: вычислительных блоков, кэш-памяти ЦПУ или буфера перевода с преобразованием (TLB).

В тех случаях, когда многопроцессорные системы включают в себя несколько полных блоков обработки, многопоточность направлена на максимизацию использования ресурсов одного ядра, используя параллелизм на уровне потоков, а также на уровне инструкций.

Поскольку эти два метода являются взаимодополняющими, их иногда объединяют в системах с несколькими многопоточными ЦП и в ЦП с несколькими многопоточными ядрами. Многопоточная парадигма стала более популярной с конца 1990-х годов, поскольку усилия по дальнейшему использованию параллелизма на уровне инструкций застопорились.

Смысл многопоточности — квазимногозадачность на уровне одного исполняемого процесса. [3]

В данной лабораторной работе рассмотриваются алгоритмы:

1) последоательный алгоритм k-means;
2) параллельный алгоритм k-means.

Цель лабораторной работы:
изучить и реализовать параллельные вычисления.

Задачи лабораторной работы:

1) изучить понятие параллельных вычислений;
2) изучить нечеткий алгоритм k-means;
3) реализовать:
    а) последоательный алгоритм k-means;
    б) параллельный алгоритм k-means.
4) провести замеры процессорного времени работы для реализованных алгоритмов.

1 Аналитический раздел

1.1 Нечёткий алгоритм k-means

Нечёткий алгоритм c-средних был предложен Джоном С. Данном в 1973 году (позднее усовершенствован Дж. Бездеком в 1981 году) как решение проблемы мягкой кластеризации, то есть присвоения каждого документа более чем одному кластеру.[4]

Как и его чёткий вариант — алгоритм k-средних – данный алгоритм, начиная с некоторого начального разбиения данных, итеративно минимизирует целевую функцию, которой является следующее выражение:

$$e_m(D,C) = \sum_{i=1}^{|D|}\sum_{j=1}^{|C|} u_{ij}^m \|\vec{d_i} - \vec{\mu_j}\|, \tag{1.1}$$

где m – степень нечёткости, $1 < m < \infty$;

u_{ij} – степень принадлежности i-ого документа j-ому кластеру,

$u_{ij} \in [0;1]$, $\sum_{j=1}^{|C|} u_{ij} = 1$ для любого $d_i \in D$;

$$u_{ij} = \frac{1}{\sum_{k=1}^{|C|} (\frac{\|\vec{d_i} - \vec{c_j}\|}{\|\vec{d_i} - \vec{c_k}\|})^{\frac{2}{m-1}}}, \tag{1.2}$$

где μ_j – центроид, кластера $C_j, j = \overline{1,|C|}$, вычисляющийся по формуле:

$$\mu_j = \frac{\sum_{i=1}^{|D|} u_{ij}^m \times \vec{d_i}}{\sum_{i=1}^{|D|} u_{ij}^m}. \tag{1.3}$$

1.1.1 Алгоритм в общем виде

Вход: массив точек в N-мерном пространстве, количество кластеров $|C| = k$.

1) Инициализация матрицы $U^0 = (u_{ij}), i = \overline{1,|D|}, j = \overline{1,k}$, например, случайными числами;
2) $t = t + 1$, где t - номер итерации;
3) Вычислить текущие центроиды кластеров $\{\vec{\mu_j}\}^t, j = \overline{1,k}$ по формуле (1.3);
4) Обновить матрицу нечёткого разбиения, то есть вычислить $U^t = \{\vec{\mu_j}\}$ по формуле (1.2);
5) Если не достигнуто условие остановки, например, $\|U^t - U^{t-1}\| < \varepsilon$, где $0 < \varepsilon < 1$, то повторить с пунта 2.

Выход: матрица степеней принадлежности точек кластерам $U^t = (u_{ij})$.

Вывод

Был рассмотрен нечёткий алгоритм k-means.

2 Констукторский раздел

В данном разделе будут рассмотрены схемы алгоритмов, требования к функциональности ПО, и определены способы тестирования.

2.1 Требования к функциональности ПО

В данной работе требуется обеспечить следующую функциональность.

1) Пользовательский режим:
    а) возможность подать на вход матрицу точек;
    б) вывод результата работы алгоритмов.
2) Тестовый режим:
    а) возможность замера процессорного времени реализации алгоритма с параллельными вычислениями с варьируемым количеством потоков.

2.2 Схемы алгоритмов

Ниже будут представлены схемы алгоритмов:

1) последовательный k-means (рисунок 2.1);
2) параллельный k-means (рисунок 2.3).

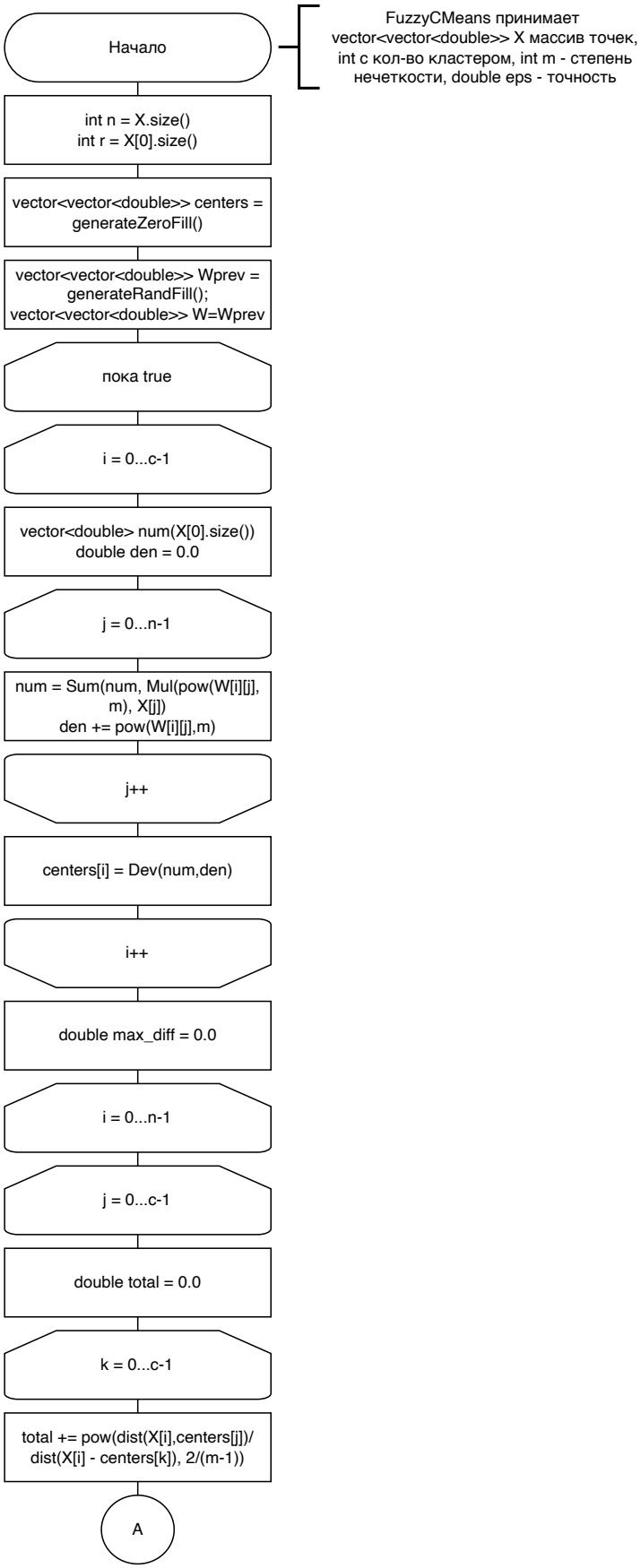

Рисунок 2.1 — Последовательный нечеткий алгоритм c-средних Часть 1

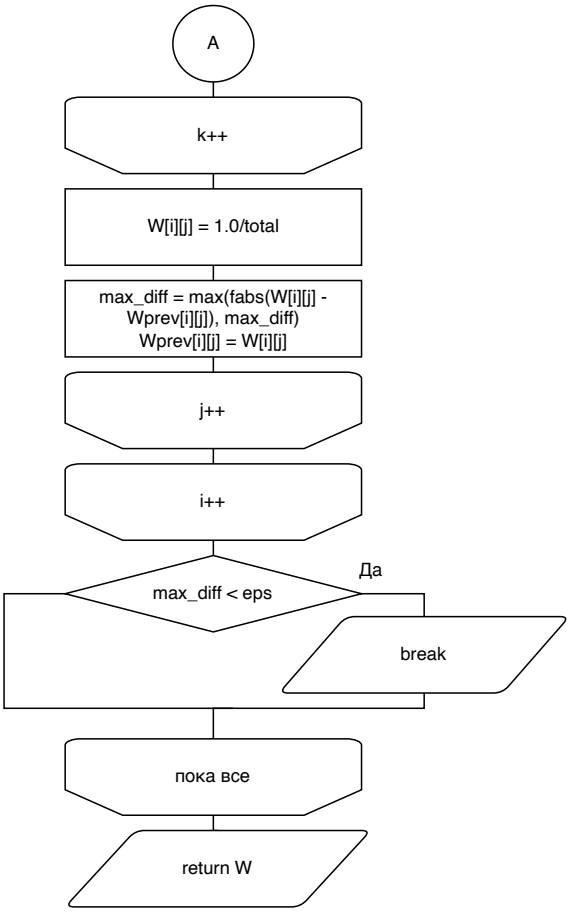

Рисунок 2.2 — Последовательный нечеткий алгоритм с-средних Часть 2

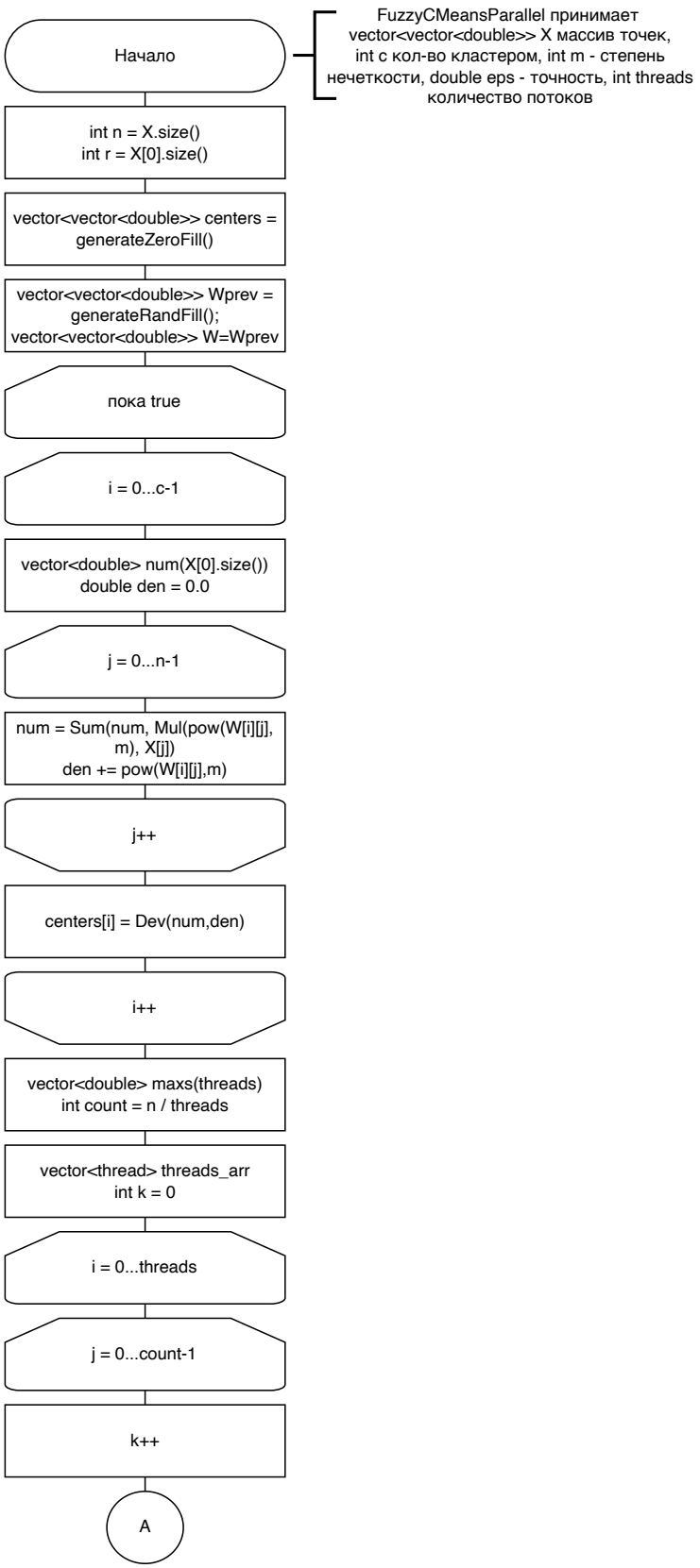

Рисунок 2.3 — Параллельный нечеткий алгоритм с-средних Часть 1

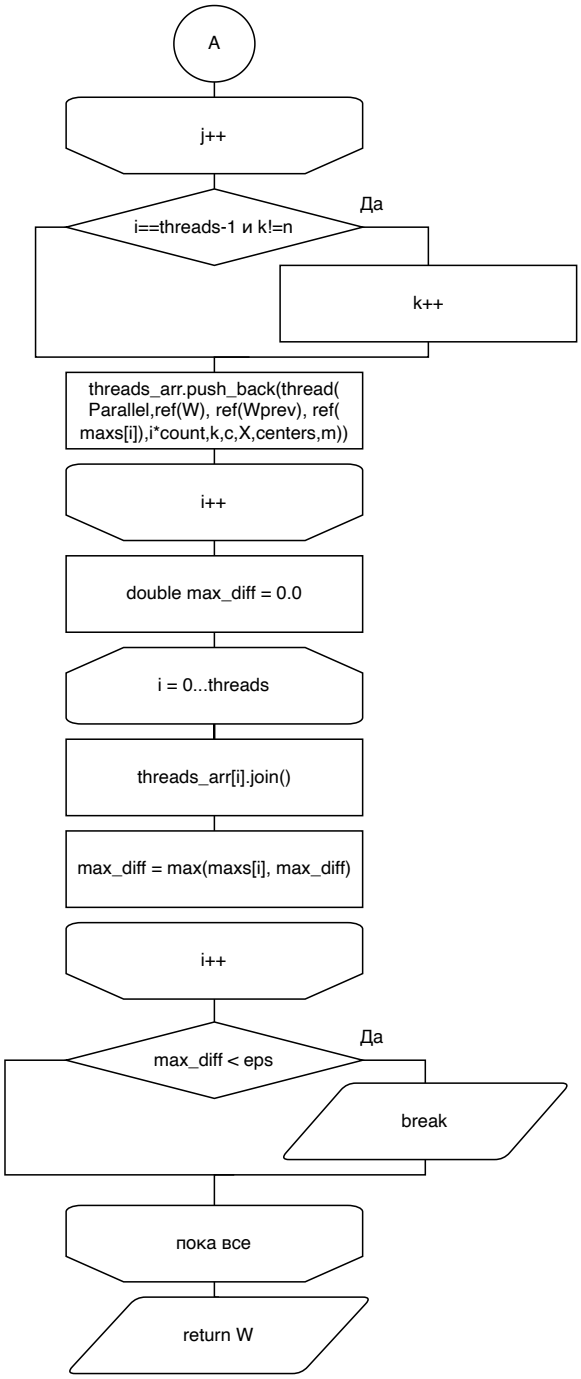

Рисунок 2.4 — Параллельный нечеткий алгоритм c-средних Часть 2

2.3 Тесты

Тестирование ПО будет проводиться методом чёрного ящика. Необходимо проверить работу системы на массивах различной длины.

Вывод

На основе теоретических данных, полученных из аналитического раздела, была построена схема нечёткого алгоритма k-means, а так же после разделения алгоритма на этапы была предложена схема параллельного выполнения данных этапов.

3 Технологический раздел

В данном разделе представлены средства, использованные в процессе разработки для реализации задачи, а также листинг кода программы. Кроме того показаны результаты тестирования и анализа затрачиваемой памяти.

3.1 Средства реализации

Для реализации поставленной задачи был использован язык C++, так как имеется большой опыт работы с ним. Для измерения процессорного времени была использована ассемблерная вставка.

3.2 Сведения о модулях программы

Программа состоит из:

— main.cpp – главный файл программы, в нем располагается точка входа;
— Alg.cpp – содержит последовательный и параллельный нечёткие алгоритмы k-means;
— time.cpp – содержит ассемблерную вставку для замера времени.

3.3 Листинг программы

В приведенных ниже листингах представлены следующие реализации:

1) последовательный нечёткий алгоритм k-means(листинг 3.1;
2) параллельный нечёткий алгоритм k-means (листинг **??**).

Листинг 3.1 — Последовательный нечёткий алгоритм k-means

```
1  vector<vector<double>> FuzzyCMeans (vector<vector<double>> X, int c, int m, double
       eps)
2  {
3      int n = X.size();
4      int r = X[0].size();
5      vector<vector<double>> centers = generateZeroFill(c, r);
6      vector<vector<double>> Wprev = generateRandFill(n, c);
7      vector<vector<double>> W = Wprev;
8      while (true){
9          for (int i=0; i<c; i++)
10         {
11             vector<double> num(X[0].size());
12             double den = 0.0;
13             for (int j=0; j<n; j++)
14             {
```

Листинг 3.2 — Последовательный нечёткий алгоритм k-means

```
1
2                   num = SumVecVec(num, MulNumVec(pow(W[j][i], m),X[j]));
3                   den += pow(W[j][i], m);
4               }
5               centers[i] = DevVecNum(num, den);
6           }
7           double max_diff = 0.0;
8           for (int i=0; i<n; i++)
9               for (int j=0; j<c; j++)
10              {
11                  double total = 0.0;
12                  for (int k=0; k<c; k++)
13                  {
14                      total += pow(distance(X[i], centers[j]) / distance(X[i],
                            centers[k]), 2/(m-1));
15                  }
16                  W[i][j] = 1.0/total;
17                  max_diff = max(fabs(W[i][j]-Wprev[i][j]), max_diff);
18                  Wprev[i][j] = W[i][j];
19              }
20          if (max_diff < eps)
21              break;
22      }
23      return W;
24  }
```

Листинг 3.3 — Параллельный нечёткий алгоритм k-means

```
1   vector<vector<double>> FuzzyCMeansParallel(vector<vector<double>> X, int c, int m,
        double eps, int threads)
2   {
3       int n = X.size();
4       int r = X[0].size();
5       vector<vector<double>> centers = generateZeroFill(c, r);
6       vector<vector<double>> Wprev = generateRandFill(n, c); //membership
7       vector<vector<double>> W = Wprev;
8       while (true){
9           // обновляем центры кластеров
10          for (int i=0; i<c; i++)
11          {
12              vector<double> num(X[0].size());
13              double den = 0.0;
14              for (int j=0; j<n; j++)
15              {
```

Листинг 3.4 — Параллельный нечёткий алгоритм k-means

```
                num = SumVecVec(num, MulNumVec(pow(W[j][i], m),X[j]));
                den += pow(W[j][i], m);
            }
            centers[i] = DevVecNum(num, den);
        }
           //обновляем вероятности W и считаем расстояние между матрицами W и Wprev
        vector<double> maxs(threads);
        int count = n / threads;
        int k = 0;
        vector<thread> threads_arr;
        for(int i = 0; i < threads; i++)
        {
            for(int j = 1; j < count+1; j++, k++)
                ;
            if (i == threads - 1 && k != n)
            {
                k++;
            }
            threads_arr.push_back(std::thread(ParallelMatrix, std::ref(W),
                std::ref(Wprev), std::ref(maxs[i]),i*count,k,c, X, centers, m));
        }
        double max_diff = 0.0;
        for(int i = 0; i < threads; i++)
        {
            threads_arr[i].join();
            max_diff = max(maxs[i], max_diff);

        }
        if (max_diff < eps)
            break;
    }
    return W;
}
```

3.4 Тестирование

В таблице 3.1 отображён возможный набор тестов для тестирования методом чёрного ящика, в соответствующих столбцах таблицы представлены результаты тестирования.

Вывод

Были разработаны и протестированы спроектированные алгоритмы и указаны средства реализации поставленной задачи.

Таблица 3.1 — Тесты проверки корректности программы

№	Input	Fuzzy	FuzzyPar
1	$\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}$	$\begin{bmatrix} 0 & nan \\ nan & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & nan \\ nan & 0 \end{bmatrix}$
2	$\begin{bmatrix} 1.4 & 1.0 & 1.3 \\ 4.3 & 4.0 & 4.8 \\ 9.5 & 9.2 & 9.1 \end{bmatrix}$	$\begin{bmatrix} 0.0221843 & 0.977816 \\ 0.208947 & 0.791053 \\ 0.999001 & 0.000998518 \end{bmatrix}$	$\begin{bmatrix} 0.0221843 & 0.977816 \\ 0.208947 & 0.791053 \\ 0.999001 & 0.000998518 \end{bmatrix}$
3	$\begin{bmatrix} 1.4 & 1.4 \\ 2.5 & 2.4 \\ 10.3 & 10.4 \\ 6.3 & 6.4 \end{bmatrix}$	$\begin{bmatrix} 0.00742315 & 0.992577 \\ 0.0043424 & 0.995658 \\ 0.968385 & 0.0316153 \\ 0.754249 & 0.245751 \end{bmatrix}$	$\begin{bmatrix} 0.00742315 & 0.992577 \\ 0.0043424 & 0.995658 \\ 0.968385 & 0.0316153 \\ 0.754249 & 0.245751 \end{bmatrix}$

4 Экспериментальный раздел

В данном разделе будут проведены эксперименты для проведения сравнительного анализа алгоритмов по затрачиваемому процессорному времени в зависимости от длины массива и степени его отсортированности.

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

В рамках данного проекта были проведёны следующие эксперименты:

1) сравнение времени работы параллельного нечёткого алгоритма k-means при разном количестве потоков(график 4.1);

2) сравнение времени работы параллельного и последовательного нечёткого алгоритма k-means при разном размере массива(график 4.2).

Тестирование проводилось на компьютере с процессором Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz под управлением Windows 11 с 8 Гб оперативной памяти.

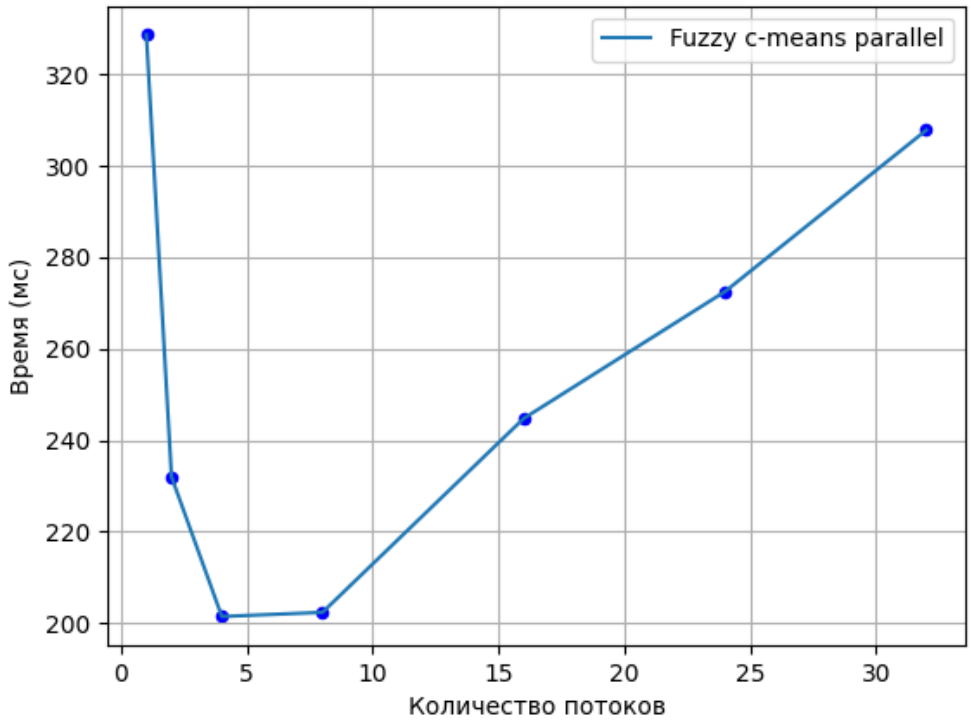

Рисунок 4.1 — Зависимость времени работы параллельного нечеткого алгоритма k-means от количества потоков

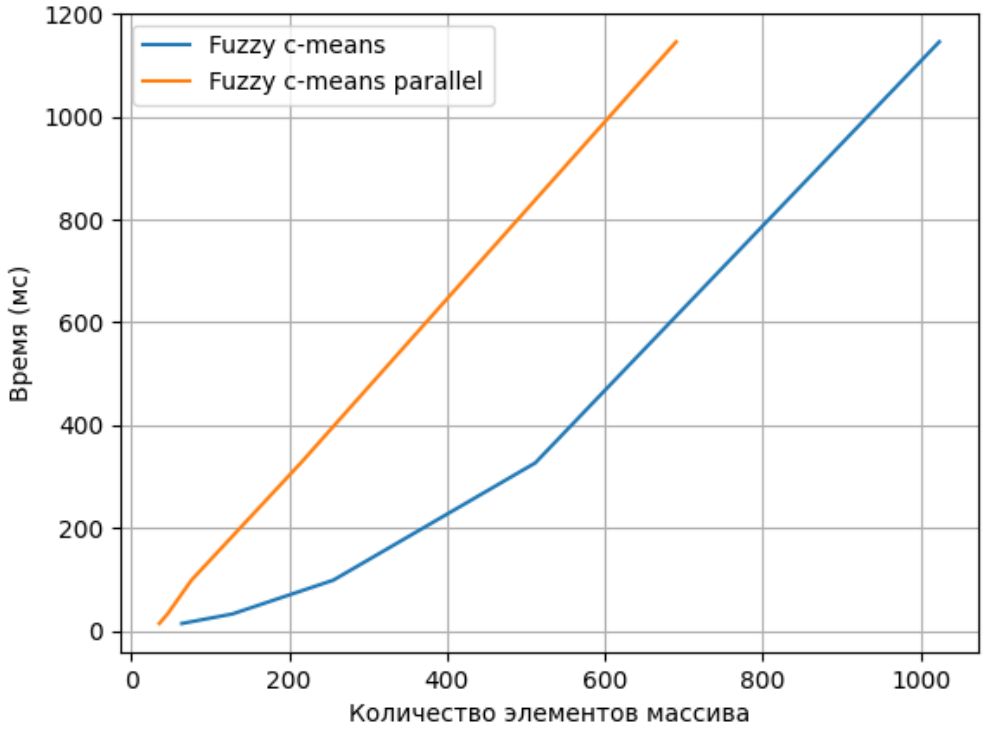

Рисунок 4.2 — Зависимость времени работы параллельного и последовательного нечеткого алгоритма k-means от размера массива (количество потоков – 4)

4.2 Вывод

В ходе экспериментов по замеру времени работы было установлено, что в параллельный нечёткий алгоритм дает наиболее быстрый результат при количестве потоков равному количеству логических ядер компьютера.

Также было установлено, что параллельный алгоритм при любом размере массива работает быстрее последовательного алгоритма.

Заключение

В рамках данной лабораторной работы была достигнута её цель: изучены паралелльные вычисления. Также выполнены следующие задачи:

— было изучено понятие параллельных вычислений;

— были реализованы обычный и параллельная реализации нечеткого алгоритма k-means;

— было произведено сравнение временных характеристик реализованных алгоритмов экспериментально.

Параллельные реализации алгоритмов выигрывают по скорости у обычной (однопоточной) реализации нечеткого алгоритма k-means. Наиболее эффективны данные алгоритмы при количестве потоков, совпадающем с количеством логических ядер компьютера. Так, например, на матрицах размером 512 на 512, удалось улучшить время выполнения нечеткого алгоритма k-means в 2.5 раза (в сравнении с однопоточной реализацией).

Список источников

1) Ассемблерные вставки в AVR-GCC. // [Электронный ресурс]. Режим доступа: https://habr.com/ru/post/275937/ дата обращения: 20.10.2022);

2) C/C++: как измерять процессорное время. // [Электронный ресурс]. Режим доступа: https://habr.com/ru/post/282301/ (дата обращения: 20.10.2022);

3) Многопоточность в C++. Основные понятия. // [Электронный ресурс]. Режим доступа: https://radioprog.ru/post/1402 (дата обращения: 20.10.2022).

4) Автоматическая обработка текстов на естественном языке и компьютерная лингвистика : учеб. пособие / Большакова Е.И., Клышинский Э.С., Ландэ Д.В., Носков А.А., Пескова О.В., Ягунова Е.В. — М.: МИЭМ, 2011. — 272 с.