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 № 5

по курсу «Анализ алгоритмов»

на тему: «Организация асинхронного взаимодействия потоков вычисления на примере конвейерных вычислений»

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

ВВЕДЕНИЕ

Целью данной лабораторной работы является описание параллельных конвейерных вычислений.

Для достижения поставленной цели необходимо решить следующие задачи:

1) описать организацию конвейерной обработки данных;

2) описать алгоритмы обработки данных, которые будут использоваться в текущей лабораторной работе;

3) определить средства программной реализации;

4) реализовать программу, выполняющую конвейерную обработку с количеством лент не менее трех в однопоточной и многопоточной реализаций;

5) сравнить и проанализировать реализации алгоритмов по затраченному времени.

1 Аналитическая часть

В данном разделе приведена информация, касающаяся основ конвейерной обработки данных.

1.1 Конвейерная обработка данных

Конвейер — организация вычислений, при которой увеличивается количество выполняемых инструкций за единицу времени за счет использования принципов параллельности.

Конвейеризация в компьютерной обработке данных основана на разбиении выполнения функций на более мелкие этапы, называемые ступенями, и выделении отдельной аппаратуры для каждой из них. Это позволяет организовать передачу данных от одного этапа к следующему, что увеличивает производительность за счет одновременного выполнения нескольких команд на различных ступенях конвейера.

Хотя конвейеризация не уменьшает время выполнения отдельной команды, она повышает пропускную способность процессора, что приводит к более быстрому выполнению программы по сравнению с простой, не конвейерной схемой.

1.2 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 [1].

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 $[0; 1]$ [1].

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)

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

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

5) если $\|U^{(t)} < U^{(t-1)}\| \geq \varepsilon$ перейти на шаг 3 [2].

1.3 Описание алгоритмов

В качестве операций, выполняющихся на конвейере, взяты следующие:

1) обработка файла, описывающего датасет файлов MIDI [3];

2) извлечение биграмм нот;

3) кластеризация нечетким алгоритмом c-средних.

Ленты конвейера (обработчики) будут передавать друг другу заявки. Первый этап, или обработчик, будет формировать заявку, которая будет передаваться от этапа к этапу. Заявка содержит:

1) массив нот, извлеченных из файла MIDI;

2) массив биграмм нот;

3) результат кластеризации: матрица разбиения и список центроидов кластеров.

Вывод

В данном разделе было рассмотрено понятие конвейерной обработки, а также выбраны этапы для обработки датасета файлов MIDI. Также был рассмотрен и описан алгоритм кластеризации с-средних.

2 Конструкторская часть

В данном разделе будут представлены схемы последовательной и параллельной работы стадий конвейера.

2.1 Требования к программному обеспечению

К программному обеспечению предъявлен ряд требований:

1) наличие интерфейса для выбора действий;

2) возможность обработки файлов MIDI;

3) возможность выбора линейной или конвейерной реализаций алгоритма.

2.2 Разработка алгоритмов

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

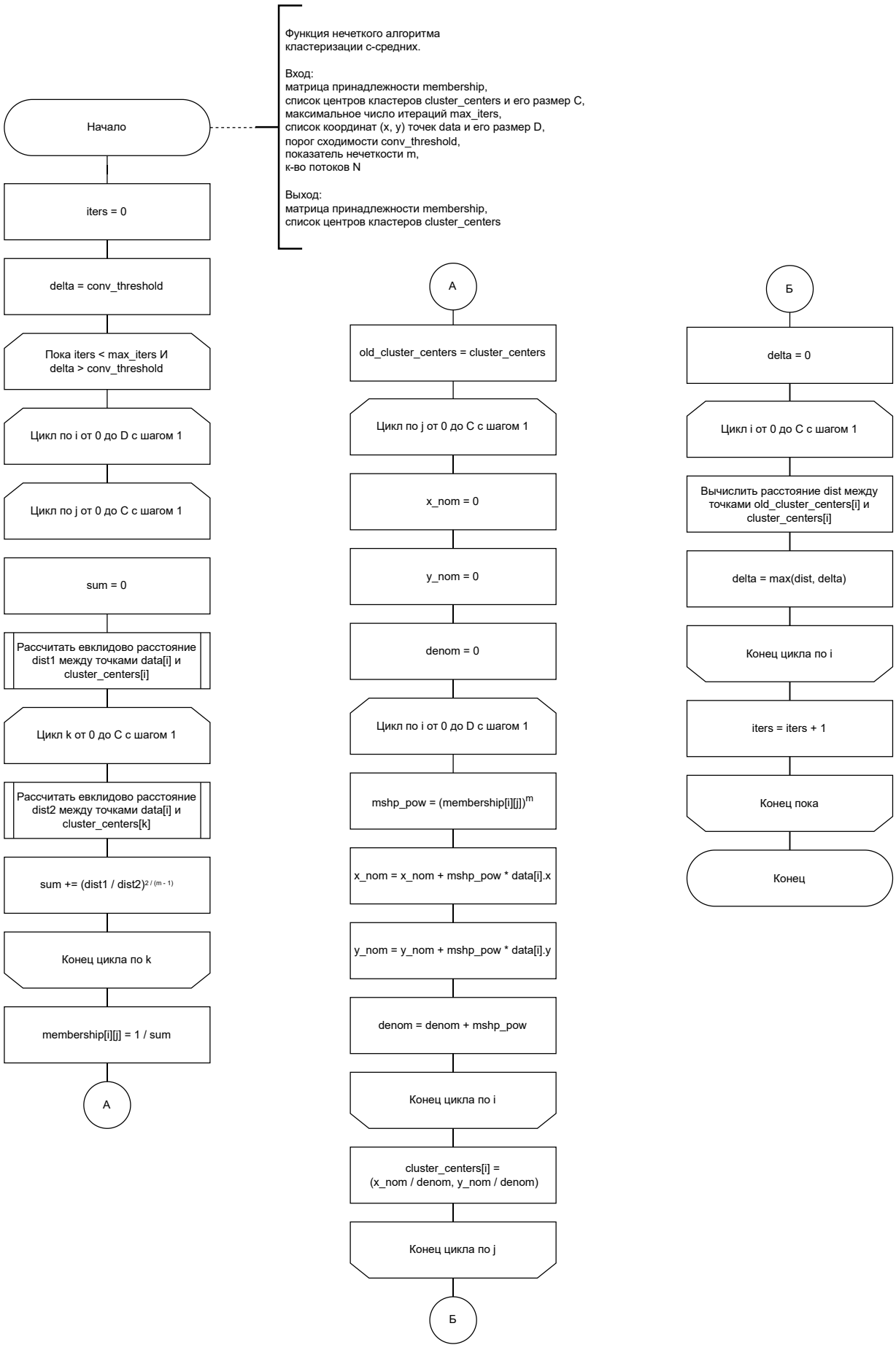

Рисунок 2.1 – Схема последовательного нечеткого алгоритма c-средних

На рисунке 2.2 представлена схема алгоритма линейной обработки датасетов файлов MIDI.

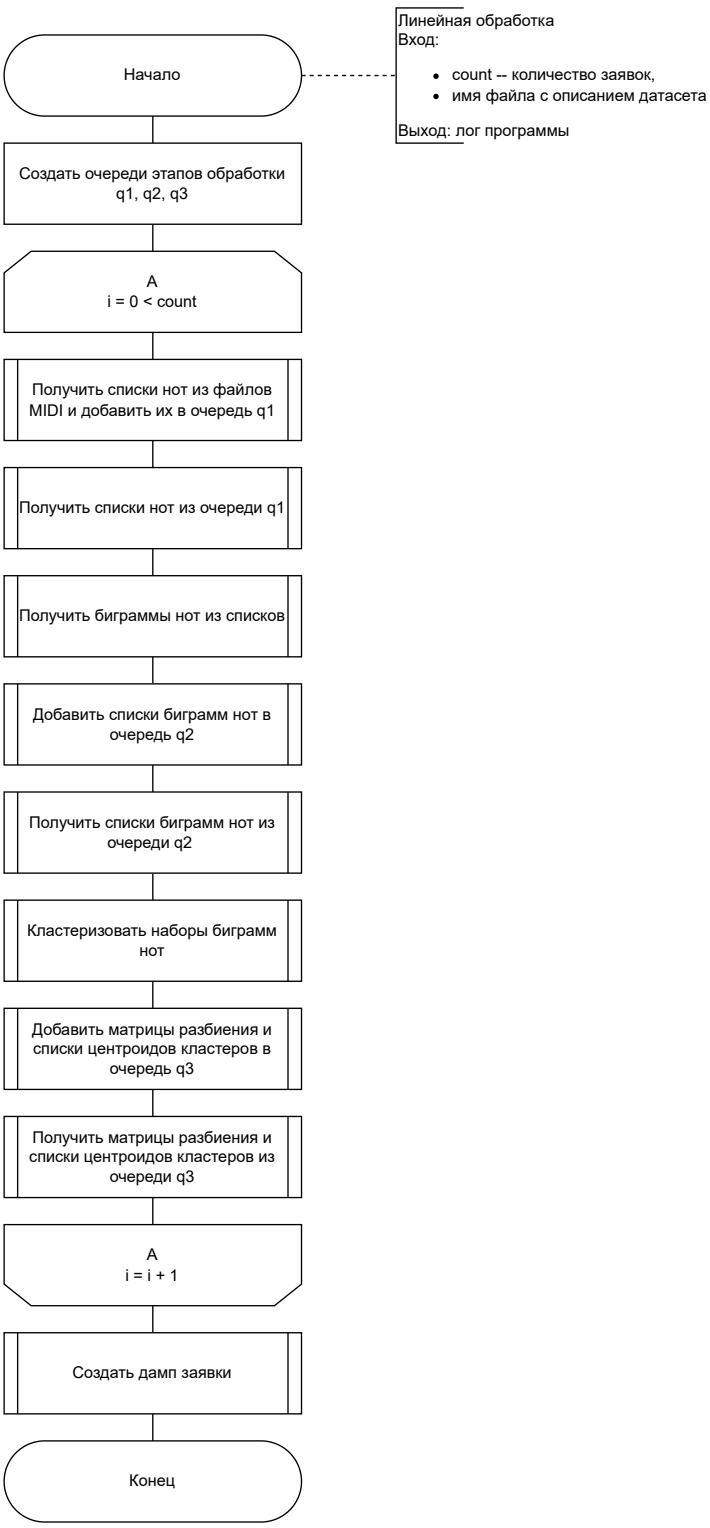

Рисунок 2.2 – Схема алгоритма линейной обработки датасетов файлов MIDI

На рисунке 2.3 представлена схема алгоритма главного потока конвейерной обработки датасетов файлов MIDI.

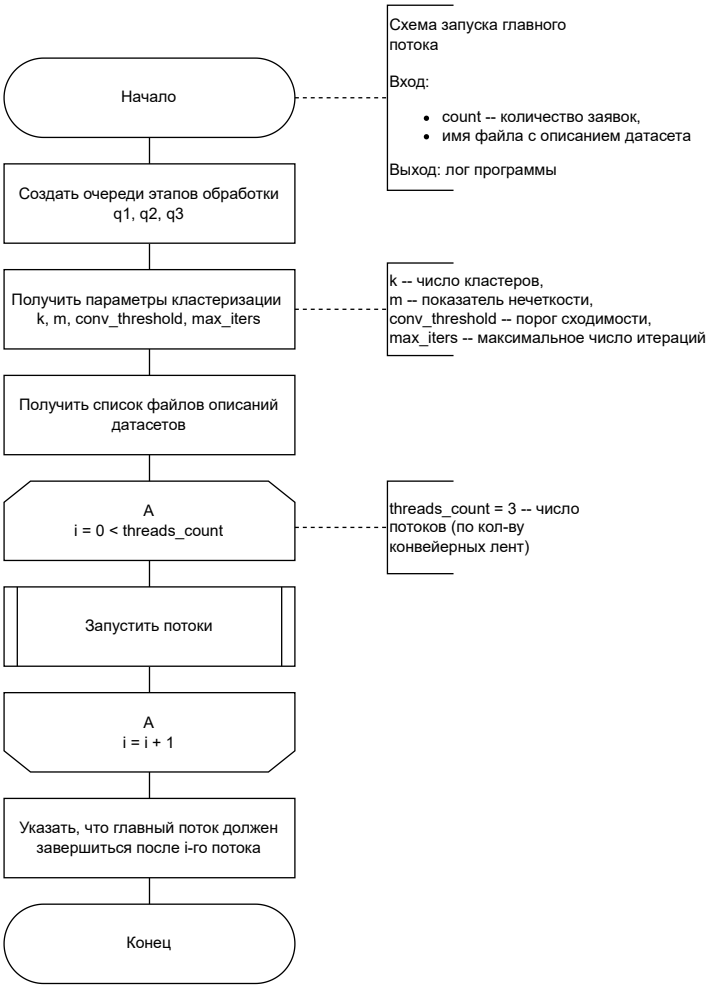

Рисунок 2.3 – Схема алгоритма главного потока конвейерной обработки датасетов файлов MIDI

На рисунке 2.4 представлена схема алгоритма потока, выполняющего извлечение нот из MIDI файлов.

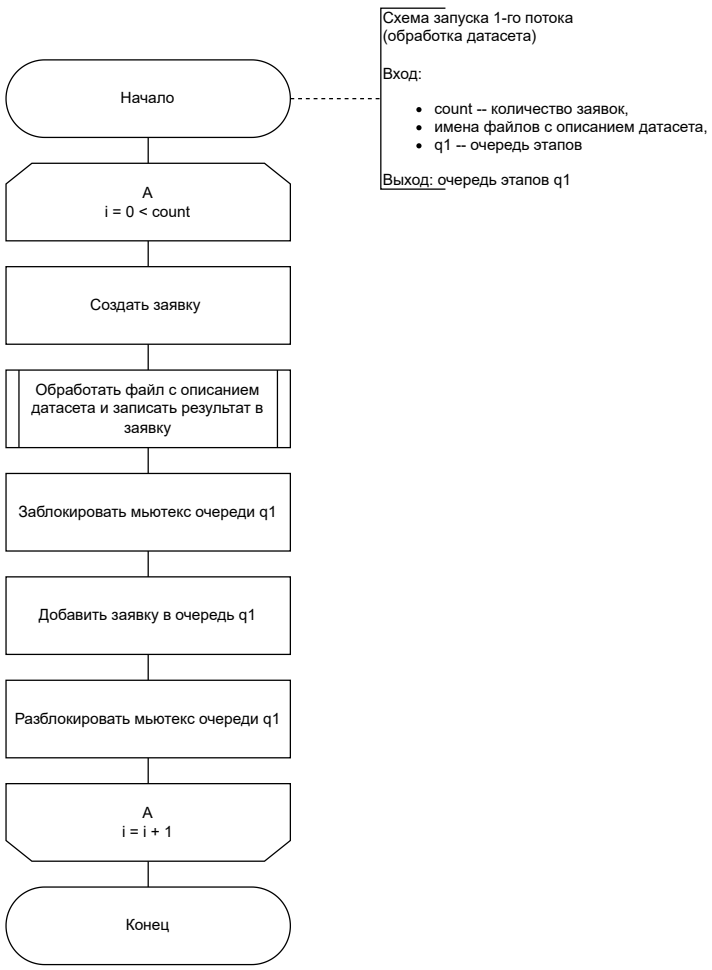

Рисунок 2.4 – Схема алгоритма потока, выполняющего извлечение нот из MIDI файлов

На рисунке 2.5 представлена схема алгоритма потока, выполняющего извлечение биграмм нот из обработанных на предыдущем этапе файлов.

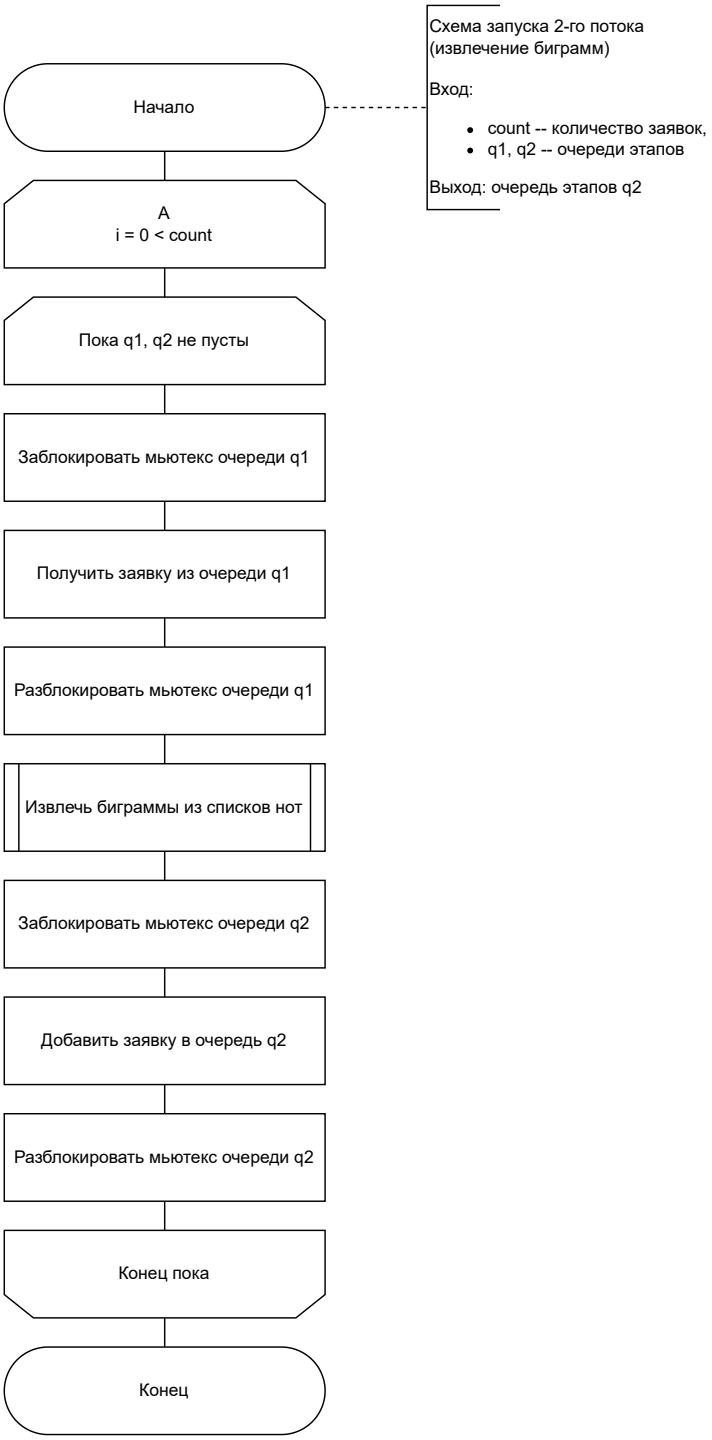

Рисунок 2.5 – Схема алгоритма потока, выполняющего извлечение биграмм нот из обработанных на предыдущем этапе файлов

На рисунке 2.6 представлена схема алгоритма потока, выполняющего кластеризацию биграмм нот.

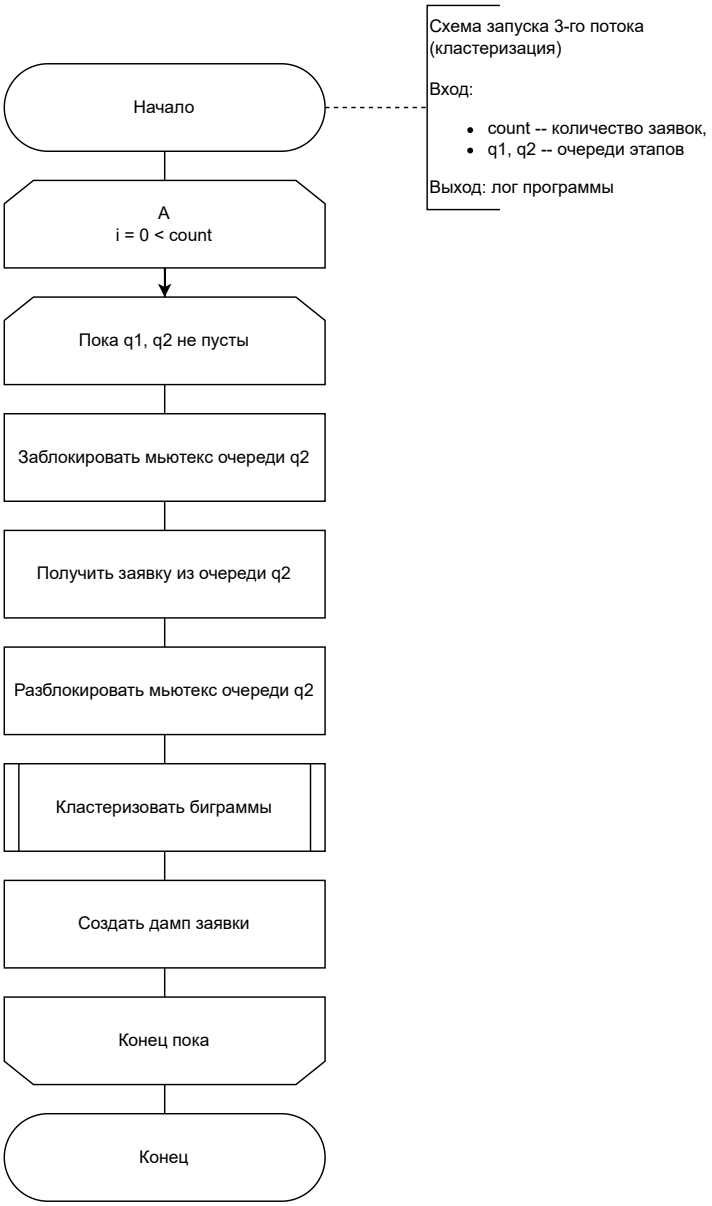

Рисунок 2.6 – Схема алгоритма потока, выполняющего кластеризацию биграмм нот

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. Также выбранный язяк программирования предоставляет возможность работы с:

— потоками (класс `thread` [6]);

— мьютексами (класс `mutex` [7]);

Обработка файлов MIDI производилась с помощью библиотеки MidiFile [8].

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

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

— `algorithms.cpp` — файл, содержащий реализации алгоритмов, используемых на различных стадиях конвейера;

— `ts_queue.cpp` — файл, содержащий реализацию потокобезопасной очереди;

— `pipeline.cpp` — файл, содержащий функции конвейерной обработки;

— `utils.cpp` — файл, содержащий вспомогательные функции;

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

3.3 Реализация алгоритмов

На листинге 3.1 представлены реализации алгоритмов извлечения нот из файлов MIDI, извлечения биграмм и кластеризации c-средних.

Листинг 3.1 – Реализации алгоритмов извлечения нот из файлов MIDI, извлечения биграмм и кластеризации c-средних

```
std::vector<note_vector_t> read_dataset(const std::string
   &descr_file)
{
    auto [id, n, filenames] = read_descr_file(descr_file);
    std::vector<note_vector_t> notes;
    for (const auto &filename : filenames)
    {
        smf::MidiFile midifile;
        midifile.read(filename);

        note_vector_t temp_notes;
        for (int track = 0; track < midifile.getTrackCount();
           track++)
        {
            for (int event = 0; event < midifile[track].size();
               event++)
            {
                if (midifile[track][event].isNoteOn())
                {
                    temp_notes.push_back(
                        midifile[track][event][1]);
                }
            }
        }
        notes.emplace_back(std::move(temp_notes));
    }
    return notes;
}

std::vector<point_vec_t> extract_bigrams(const
   std::vector<note_vector_t> &notes)
{
    std::vector<point_vec_t> bigrams;
    for (const auto &note_set : notes)
    {
```

```
32          point_vec_t temp_bigrams;
33          for (size_t i = 1; i < note_set.size(); ++i)
34          {
35              std::vector<double> bigram{
36                  (double)note_set[i - 1], (double)note_set[i]};
37              temp_bigrams.emplace_back(bigram);
38          }
39          bigrams.emplace_back(temp_bigrams);
40      }
41      return bigrams;
42  }
43
44  void c_means(
45          membership_t &membership, point_vec_t &cluster_centers,
46          const point_vec_t &data,
47          double m, double conv_threshold, int max_iters)
48  {
49      int iters = 0;
50      double delta = conv_threshold + 1.0;
51      while (iters < max_iters && delta > conv_threshold)
52      {
53          for (size_t i = 0; i < data.size(); ++i)
54          {
55              for (size_t j = 0; j < cluster_centers.size(); ++j)
56              {
57                  double sum = 0.0;
58                  double dist1 = sqrt(pow(data[i][0] -
                        cluster_centers[j][0], 2) +
59                                      pow(data[i][1] -
                                          cluster_centers[j][1],
                                          2));
60                  for (size_t k = 0; k < cluster_centers.size();
                        ++k)
61                  {
62                      double dist2 = sqrt(pow(data[i][0] -
                            cluster_centers[k][0], 2) +
63                                          pow(data[i][1] -
                                              cluster_centers[k][1],
                                              2));
64                      sum += pow(dist1 / dist2, 2.0 / (m - 1.0));
65                  }
```

```
66                  membership[i][j] = 1.0 / sum;
67              }
68          }
69          auto old_cluster_centers = cluster_centers;
70          for (size_t j = 0; j < cluster_centers.size(); ++j)
71          {
72              double x_nom = 0.0, y_nom = 0.0, denom = 0.0;
73              for (size_t i = 0; i < data.size(); ++i)
74              {
75                  double membership_pow_m = pow(membership[i][j],
                        m);
76                  x_nom += membership_pow_m * data[i][0];
77                  y_nom += membership_pow_m * data[i][1];
78                  denom += membership_pow_m;
79              }
80              cluster_centers[j] = {x_nom / denom, y_nom / denom};
81          }
82          delta = 0.0;
83          for (size_t i = 0; i < cluster_centers.size(); ++i)
84          {
85              double distance = sqrt(pow(old_cluster_centers[i][0]
                    - cluster_centers[i][0], 2) +
86                                     pow(old_cluster_centers[i][1]
                                        - cluster_centers[i][1],
                                        2));
87              if (distance > delta)
88                  delta = distance;
89          }
90          ++iters;
91      }
92  }
```

На листинге 3.2 представлена реализация линейного алгоритма обработки набора файлов MIDI.

Листинг 3.2 – Реализация линейного алгоритма обработки набора файлов MIDI

```
1  void consequent()
2  {
3      int req_cnt = utils::get_request_count();
4      auto datasets = utils::pick_datasets(req_cnt);
5      auto [k, m, conv_threshold, max_iters] =
```

```
        utils::get_clust_params();

    std::vector<std::unique_ptr<stages_t>> pool;
    for (int i = 0; i < req_cnt; ++i)
    {
        stages_t *s = new stages_t;

        clock_gettime(CLOCK_REALTIME, &s->parsed.op_start);
        s->parsed.notes = read_dataset(datasets[i]);
        clock_gettime(CLOCK_REALTIME, &s->parsed.op_end);

        clock_gettime(CLOCK_REALTIME, &s->embedded.op_start);
        s->embedded.embeddings =
            extract_bigrams(s->parsed.notes);
        clock_gettime(CLOCK_REALTIME, &s->embedded.op_end);

        clock_gettime(CLOCK_REALTIME, &s->clusterized.op_start);
        for (const auto &embed : s->embedded.embeddings)
        {
            auto [mshp, cc] = init_structures(embed.size(), k);
            c_means(mshp, cc, embed, m, conv_threshold,
                max_iters);
            s->clusterized.results.emplace_back(std::move(mshp),
                std::move(cc));
        }
        clock_gettime(CLOCK_REALTIME, &s->clusterized.op_end);
        pool.emplace_back(s);
    }
    dump_pool(pool, "cons.txt");
}
```

На листинге 3.3 представлена реализация конвейерного алгоритма обработки набора файлов MIDI.

Листинг 3.3 – Реализация конвейерного алгоритма обработки набора файлов MIDI

```
static void service_01(
    int req_cnt,
    const std::vector<std::string> &datasets,
    ts_queue<stages_t *> &q1)
{
    for (int i = 0; i < req_cnt; ++i)
```

```
    {
        stages_t *s = new stages_t();
        clock_gettime(CLOCK_REALTIME, &s->parsed.op_start);
        s->parsed.notes = read_dataset(datasets[i]);
        clock_gettime(CLOCK_REALTIME, &s->parsed.op_end);
        q1.push(s);
    }
}

static void service_02(
    int req_cnt,
    ts_queue<stages_t *> &q1,
    ts_queue<stages_t *> &q2)
{
    for (int i = 0; i < req_cnt; ++i)
    {
        stages_t *s = q1.pop();

        clock_gettime(CLOCK_REALTIME, &s->embedded.op_start);
        s->embedded.embeddings =
           extract_bigrams(s->parsed.notes);
        clock_gettime(CLOCK_REALTIME, &s->embedded.op_end);
        q2.push(s);
    }
}

static void service_03(
    int req_cnt,
    std::tuple<int, double, double, int> cls_params,
    ts_queue<stages_t *> &q2,
    std::vector<std::unique_ptr<stages_t>> &pool)
{
    for (int i = 0; i < req_cnt; ++i)
    {
        stages_t *s = q2.pop();
        auto [k, m, conv_threshold, max_iters] = cls_params;

        clock_gettime(CLOCK_REALTIME, &s->clusterized.op_start);
        for (const auto &embed : s->embedded.embeddings)
        {
            auto [mshp, cc] = init_structures(embed.size(), k);
```

```
47              c_means(mshp, cc, embed, m, conv_threshold,
                    max_iters);
48              s->clusterized.results.emplace_back(std::move(mshp),
                    std::move(cc));
49          }
50          clock_gettime(CLOCK_REALTIME, &s->clusterized.op_end);
51          pool.emplace_back(s);
52      }
53  }
54
55  void concurrent()
56  {
57      int req_cnt = utils::get_request_count();
58      auto datasets = utils::pick_datasets(req_cnt);
59      std::tuple cls_params = utils::get_clust_params();
60
61      std::vector<std::unique_ptr<stages_t>> pool;
62      ts_queue<stages_t *> q1;
63      ts_queue<stages_t *> q2;
64
65      std::thread t_01(service_01, req_cnt, std::cref(datasets),
            std::ref(q1));
66      std::thread t_02(service_02, req_cnt, std::ref(q1),
            std::ref(q2));
67      std::thread t_03(service_03, req_cnt, cls_params,
            std::ref(q2), std::ref(pool));
68
69      t_01.join();
70      t_02.join();
71      t_03.join();
72
73      dump_pool(pool, "conc.txt");
74  }
```

Вывод

В данном разделе были рассмотрены средства реализации, а также представлен листинг реализаций линейного и конвейерного алгоритмов обработки набора файлов MIDI.

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 2 «Запустить последовательную обработку датасета файлов MIDI», выбрал файл с описанием датасета под номером 1 и ввел следующие значения:

— количество заявок — 10,

— число кластеров — 3,

— значение показателя нечеткости — 2,

— значение порога сходимости — 1,

— максимальное к-во итераций — 10.

```
                Меню
1. Запустить последовательную обработку датасета файлов MIDI.
2. Запустить конвейерную обработку датасета файлов MIDI.
3. Произвести замеры по времени реализуемых алгоритмов.
0. Выход.

Выберите опцию (0-3): 2

Введите количество заявок: 10
Выберите файлы описания датасета:
    1. "/home/daria/Документы/bmstu-aa/lab5/prog/descriptions/ds_02.txt"
    2. "/home/daria/Документы/bmstu-aa/lab5/prog/descriptions/ds_01.txt"
Номера файлов (0 для выхода): 2 1 2 2 1 2 2 2 1 2
Введите количество кластеров: 3
Введите показатель нечеткости: 2
Введите порог сходимости: 1
Введите максимальное количество итераций: 10

                Меню
1. Запустить последовательную обработку датасета файлов MIDI.
2. Запустить конвейерную обработку датасета файлов MIDI.
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)

Наборы данных генерировались из равномерного распределения.

4.4 Вывод

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

1)

ЗАКЛЮЧЕНИЕ

В результате выполнения лаботаторной 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 версии алгоритма;
5) проведен сравнительный анализ процессорного времени выполнения реализованных алгоритмов:
   - при варьировании размера датасета последовательная версия нечеткого алгоритма кластеризации с-средних выполнялась в среднем в 92.6 раз дольше, чем параллельная;
   - при варьировании числа потоков последовательная версия нечеткого алгоритма кластеризации с-средних выполнялась в среднем 72.7 раз дольше, чем параллельная;
   - 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.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *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.
2. *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.
3. MIDI — Wikipedia, The Free Encyclopedia. — [Электронный ресурс]. — Режим доступа: https://en.wikipedia.org/w/index.php?title=MIDI&oldid=1190344258 (дата обращения: 21.12.2023).
4. C++ language. — [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/language (дата обращения: 21.12.2023).
5. Standard library header <ctime>. — [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/thread/thread (дата обращения: 21.12.2023).
6. std::thread. — [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/thread/thread (дата обращения: 22.12.2023).
7. std::mutex. — [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/thread/mutex (дата обращения: 22.12.2023).
8. Midifile: C++ MIDI file parsing library. — [Электронный ресурс]. — Режим доступа: https://github.com/craigsapp/midifile (дата обращения: 22.12.2023).