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"

Тема Конвейер

Студент Романов С.К.

Группа ИУ7-55Б

Оценка (баллы)

Преподаватели Волкова Л.Л., Строганов Ю.В.

Москва — 2023 г.

Оглавление

Введение

Конвейер — способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности (увеличения числа инструкций, выполняемых в единицу времени — эксплуатация параллелизма на уровне инструкций), технология, используемая при разработке компьютеров и других цифровых электронных устройств.

Сам термин «конвейер» пришёл из промышленности, где используется подобный принцип работы — материал автоматически подтягивается по ленте конвейера к рабочему, который осуществляет с ним необходимые действия, следующий за ним рабочий выполняет свои функции над получившейся заготовкой, следующий делает ещё что-то. Таким образом, к концу конвейера цепочка рабочих полностью выполняет все поставленные задачи, сохраняя высокий темп производства. Например, если на самую медленную операцию затрачивается одна минута, то каждая деталь будет сходить с конвейера через одну минуту. В процессорах роль рабочих исполняют функциональные модули, входящие в состав процессора.

Цель данной работы: получить навык организации асинхронного взаимодействия потоков на примере конвейерной обработки данных.

В рамках выполнения работы необходимо решить следующие задачи:

- рассмотреть и изучить конвейерную обработку данных;
- реализовать конвейер с количеством лент не меньше трех в многопоточной среде;
- на основании проделанной работы сделать выводы.

1 Аналитическая часть

1.1 Описание конвейерной обработки данных

Конвейер[1] — способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности (увеличения числа инструкций, выполняемых в единицу времени — эксплуатация параллелизма на уровне инструкций), технология, используемая при разработке компьютеров и других цифровых электронных устройств.

Идея заключается в параллельном выполнении нескольких инструкций процессора. Сложные инструкции процессора представляются в виде последовательности более простых стадий. Вместо выполнения инструкций последовательно (ожидания завершения конца одной инструкции и перехода к следующей), следующая инструкция может выполняться через несколько стадий выполнения первой инструкции. Это позволяет управляющим цепям процессора получать инструкции со скоростью самой медленной стадии обработки, однако при этом намного быстрее, чем при выполнении эксклюзивной полной обработки каждой инструкции от начала до конца.

Многие современные процессоры управляются тактовым генератором. Процессор внутри состоит из логических элементов и ячеек памяти — триггеров. Когда приходит сигнал от тактового генератора, триггеры приобретают своё новое значение, и «логике» требуется некоторое время для декодирования новых значений. Затем приходит следующий сигнал от тактового генератора, триггеры принимают новые значения, и так далее. Разбивая последовательности логических элементов на более короткие и помещая триггеры между этими короткими последовательностями, уменьшают время, необходимое логике для обработки сигналов. В этом случае длительность одного такта процессора может быть соответственно уменьшена.

1.2 Алгоритм DBSCAN

Алгоритм, который был выбран для разложения на части конвейера – DBSCAN (Density-Based Spatial Clustering of Applications with Noise)[2][3], алгоритм кластеризации, который основывается на понятии плотности. Он ищет группы точек, которые тесно связаны друг с другом и отделены друг от друга более редко встречающимися точками.

Вывод

В данной работе стоит задача реализации конвейера для алгоритма DBSCAN.

2 Конструкторская часть

2.1 Разработка конвейерной обработки данных

Принцип работы конвейера с 3 лентами представлен на рисунке 2.1, а имплементация конвейерного решения DBSCAN на рисунке 2.2. Алгоритм DBSCAN и стадии его обработки представлены на рисунках 2.3 и 2.4.

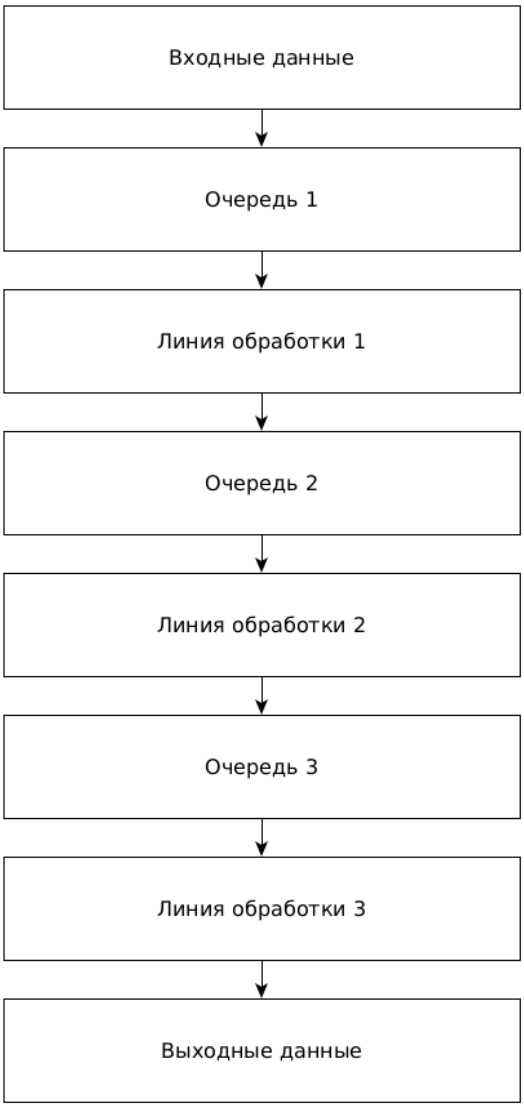

Рис. 2.1: Принцип работы конвейера с 3 лентами.

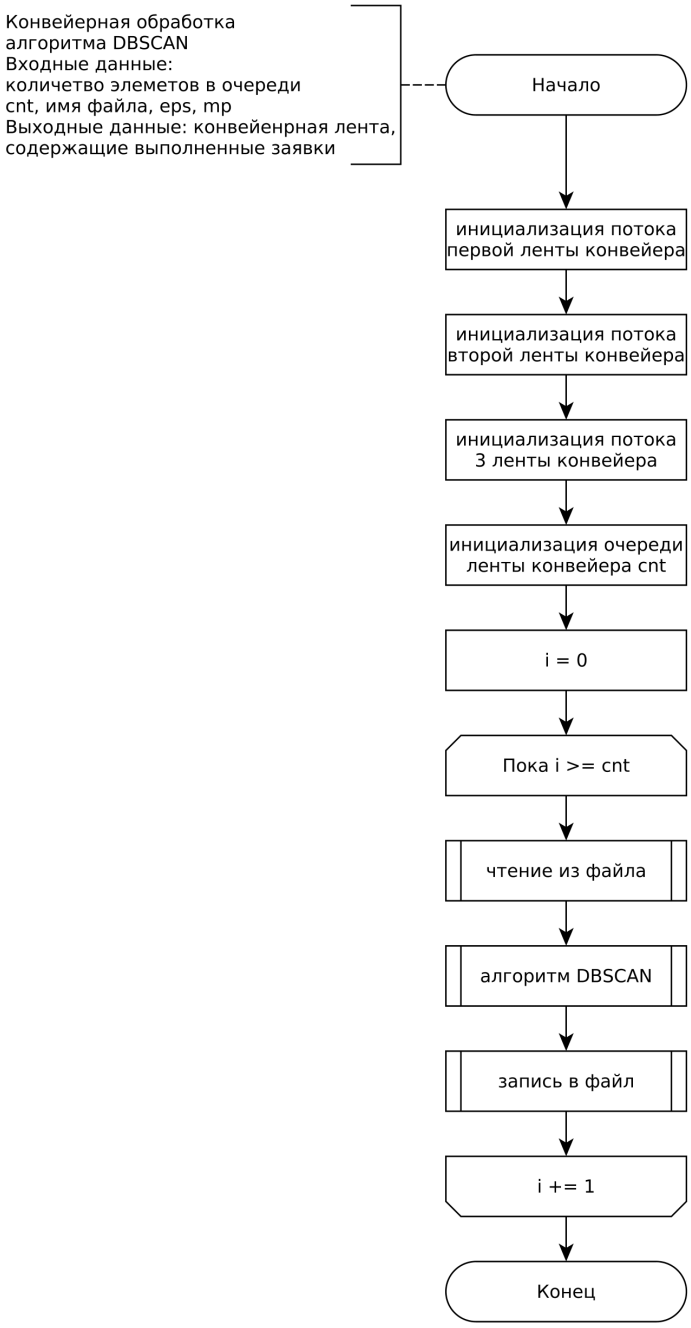

Рис. 2.2: Алгоритм конвейерного решения DBSCAN

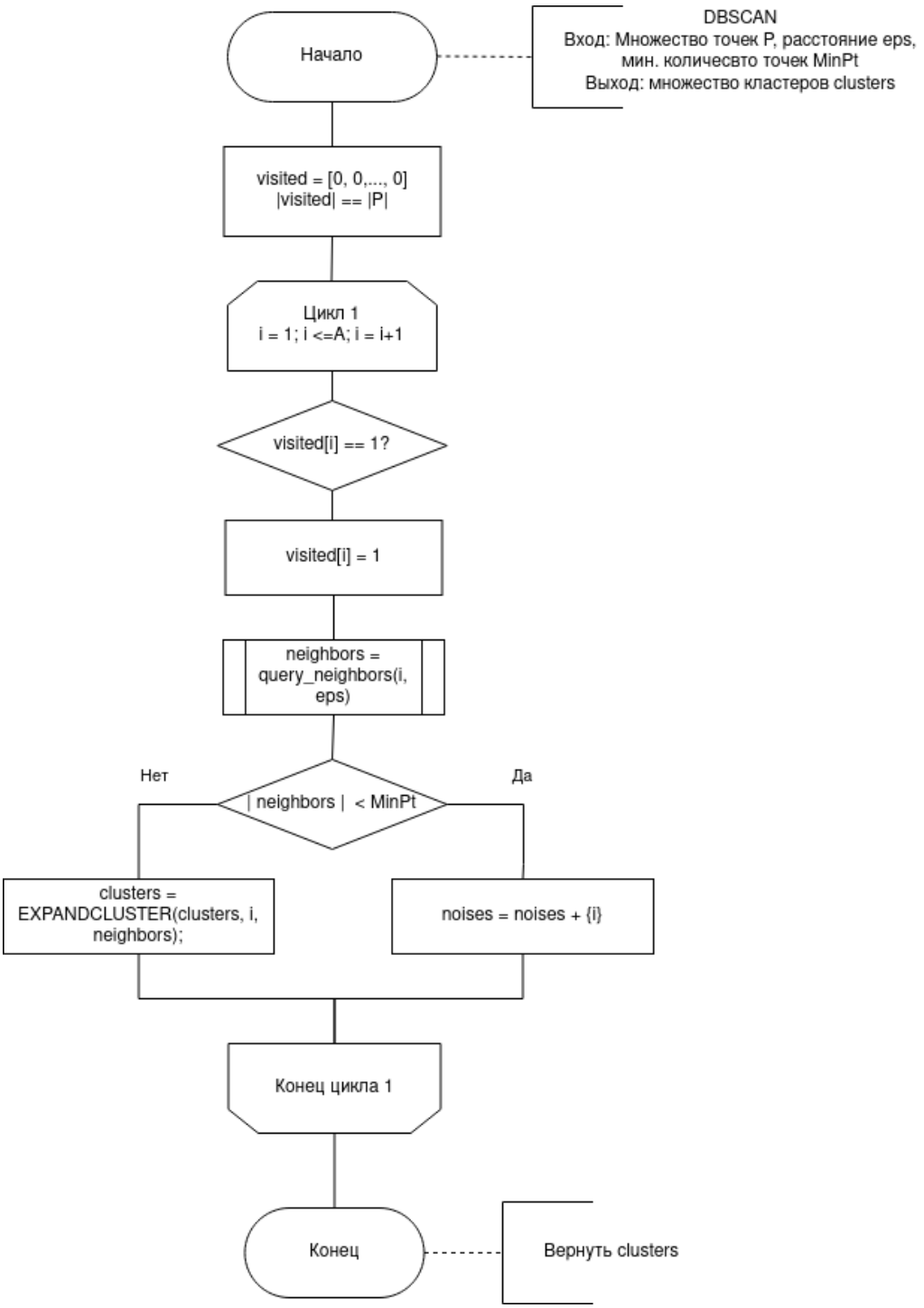

Рис. 2.3: Линейный алгоритм DBSCAN

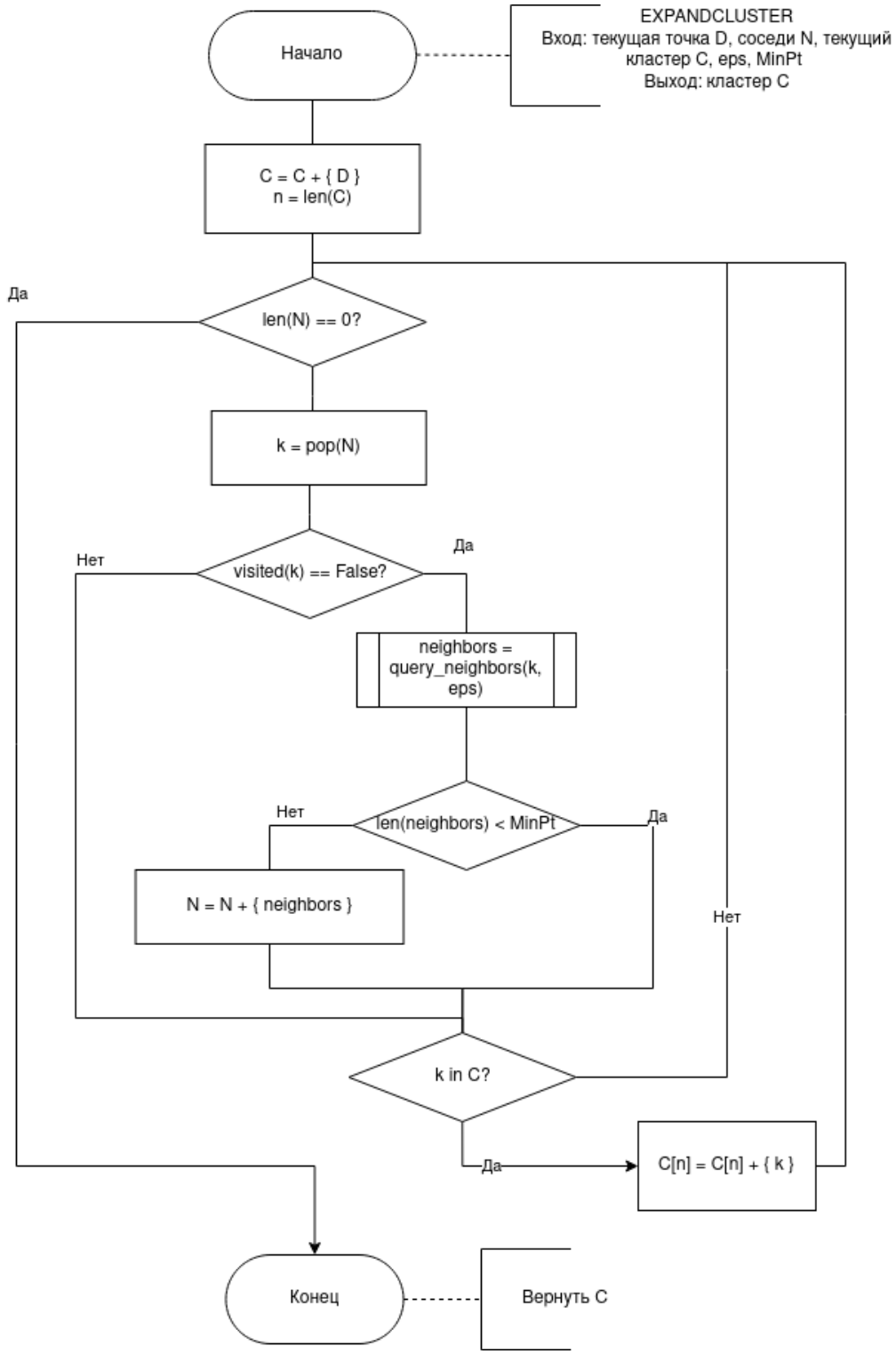

Рис. 2.4: Функция EXPANDCLUSTERS линейного алгоритма DBSCAN

Вывод

Был показан принцип работы конвейерной обработки данных, а также алгоритм Рабина-Карпа с разделениями его на этапы для возможности выполнения на конвейере.

3 Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации и сами реализации алгоритмов.

3.1 Требования к программному обеспечению

К программе предъявляется ряд условий:

– на вход ковейера подаётся массив задач, которые на нём нужно обработать;

– на выходе - лог-запись, в которой записаны в упорядоченном по времени порядке события начала и конца обработки определённого задания на ленте.

– ПО должно замерять время работы алгоритмов.

3.2 Средства реализации

Для реализации данной лабораторной работы необходимо установить следующее программное обеспечение:

– Rust Programming Language v1.64.0 - язык программирования[4]

– Criterion.rs v0.4.0 - Средство визуализации данных

– LaTeX - система верстки документов

3.3 Листинг кода

В листингах 3.1 и 3.2 приведена реализация конвейера. В листингах 3.3, 3.4, 3.5 и 3.6 приведена реализация задачи.

```rust
use std::sync::mpsc::{channel, Receiver, Sender};
use std::thread;

use super::task::DBSCANTaskResult;
pub use super::task::{DBSCANTask, ThreeTask};

pub struct Conveyor3
{
    output: Receiver<DBSCANTask>,
}

impl Conveyor3
{
    pub fn new(queue: Vec<DBSCANTask>) -> Self
    {
        let (task_input, part1_receiver): (Sender<DBSCANTask>, Receiver<DBSCANTask>) =
            channel();
        let (part1_sender, part2_receiver): (Sender<DBSCANTask>, Receiver<DBSCANTask>) =
            channel();
        let (part2_sender, part3_receiver): (Sender<DBSCANTask>, Receiver<DBSCANTask>) =
            channel();
        let (part3_sender, task_output): (Sender<DBSCANTask>, Receiver<DBSCANTask>) =
            channel();

        thread::spawn(move || {
            for mut task in part1_receiver.iter() {
                task.part_a();
                part1_sender.send(task).expect("Send␣task␣to␣part2");
            }
        });

        thread::spawn(move || {
            for mut task in part2_receiver.iter() {
                task.part_b();
                part2_sender.send(task).expect("Send␣task␣to␣part3");
            }
        });

        thread::spawn(move || {
            for mut task in part3_receiver.iter() {
                task.part_c();
                part3_sender.send(task).expect("Send␣task␣to␣output");
            }
```

Листинг 3.1: Реализация конвейера, часть 1

```
        for task in queue {
            task_input.send(task).expect("Send task to conveyor");
        }

        Self {
            output: task_output,
        }
    }

    pub fn recv(&self) -> Option<DBSCANTaskResult>
    {
        self.output.recv().map(|res| res.result()).ok()
    }
}
```

Листинг 3.2: Реализация конвейера, часть 2

```
pub trait ThreeTask<T>
{
    fn part_a(&mut self);
    fn part_b(&mut self);
    fn part_c(&mut self);

    fn run_a(&mut self);
    fn run_b(&mut self);
    fn run_c(&mut self);

    fn result(&self) -> T;
}

#[derive(Debug, Clone)]
pub struct DBSCANTask
{
    filename: String,
    model: Option<ParallelModel>,
    eps: f64,
    min_pts: usize,
    focus: Focus,
    output_file: Option<String>,
    // result: Option<Vec<usize>>,
    times: [DateTime<Utc>; NUMBER_OF_MEASURMENTS],
}
```

Листинг 3.3: Структура задачи, часть 1

```rust
impl DBSCANTask
{
    pub fn new(string: String, eps: f64, min_pts: usize, focus: (usize, usize)) -> Self
    {
        let current_time = Utc::now();
        Self {
            filename: string,
            model: None,
            eps,
            min_pts,
            focus: Focus(focus.0, focus.1),
            output_file: None,
            times: [current_time; NUMBER_OF_MEASURMENTS],
        }
    }

    fn input(&mut self)
    {
        self.model = Some(ParallelModel::new(
            Arc::new(Mutex::new(Dataset::input_dataset(
                self.filename.clone(),
                vec![self.focus.0, self.focus.1],
                b',',
            ))),
            2,
        ));
        self.model.as_mut().unwrap().set_eps(self.eps);
        self.model.as_mut().unwrap().set_min_pts(self.min_pts);
    }

    fn process(&mut self)
    {
        self.model.as_mut().unwrap().run();
    }
```

Листинг 3.4: Структура задачи, часть 2

```rust
    fn save(&mut self)
    {
        let result = self.model.as_ref().unwrap().get_clusters();
        let filename_save = format!(
            "./results/clusters_of_{}_{}_{}.csv",
            self.eps, self.min_pts, self.times[0]
        );
        let mut file = File::create(filename_save.clone()).unwrap();
        let strings: Vec<String> = result.iter().map(|n| n.to_string()).collect();
        file.write_all(strings.join(",␣").as_bytes()).unwrap();
        self.output_file = Some(filename_save);
    }

    pub fn run_all(&mut self)
    {
        self.input();
        self.process();
        self.save();
    }

    pub fn reset(&mut self)
    {
        self.model.as_mut().unwrap().reset();
    }
}

#[derive(Debug)]
pub struct DBSCANTaskResult
{
    pub result: Vec<i32>,
    pub times: [DateTime<Utc>; NUMBER_OF_MEASURMENTS],
}
```

Листинг 3.5: Структура задачи, часть 3

```rust
impl ThreeTask<DBSCANTaskResult> for DBSCANTask
{
    fn part_a(&mut self)
    {
        self.times[T1_START] = Utc::now();
        self.run_a();
        self.times[T1_END] = Utc::now();
    }

    fn part_b(&mut self)
    {
        self.times[T2_START] = Utc::now();
        self.run_b();
        self.times[T2_END] = Utc::now();
    }

    fn part_c(&mut self)
    {
        self.times[T3_START] = Utc::now();
        self.run_c();
        self.times[T3_END] = Utc::now();
    }

    fn run_a(&mut self)
    {
        self.input();
    }

    fn run_b(&mut self)
    {
        self.process();
    }

    fn run_c(&mut self)
    {
        self.save();
    }

    fn result(&self) -> DBSCANTaskResult
    {
        DBSCANTaskResult {
            result: self.model.as_ref().unwrap().get_clusters().clone(),
            times: self.times.clone(),
        }
    }
}
```

Листинг 3.6: Структура задачи, часть 4

3.4 Тестирование функций.

Одним из способов протестировать работу DBSCAN является графическое отображение полученных кластеров.

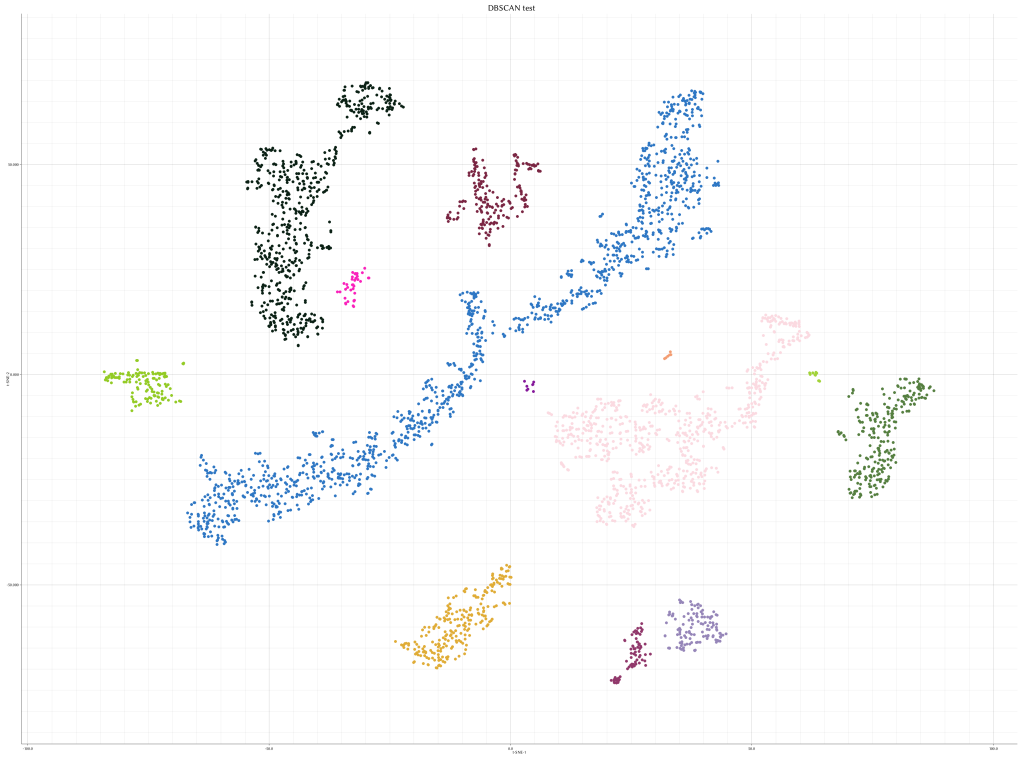

Рис. 3.1: Тестирование работы DBSCAN

Алгоритм успешно кластеризует данные.

Вывод

Была разработана реализация конвейерных вычислений.

4 Исследовательская часть

Ниже приведены технические характеристики устройства, на котором было проведено тестирование ПО:

– Операционная система: Arch Linux [5] 64-бит.

– Оперативная память: 16 Гб DDR4.

– Процессор: 11th Gen Intel® Core™ i5-11320H @ 3.20 ГГц[6].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, окружением, а также непосредственно системой тестирования.

4.1 Сравнение параллельного и последовательного конвейера

В рисунке 4.1 приведено сравнение времени выполнения линейного и параллельногоё конвейеров в зависимости от длины очереди (при количестве 4407 точек).

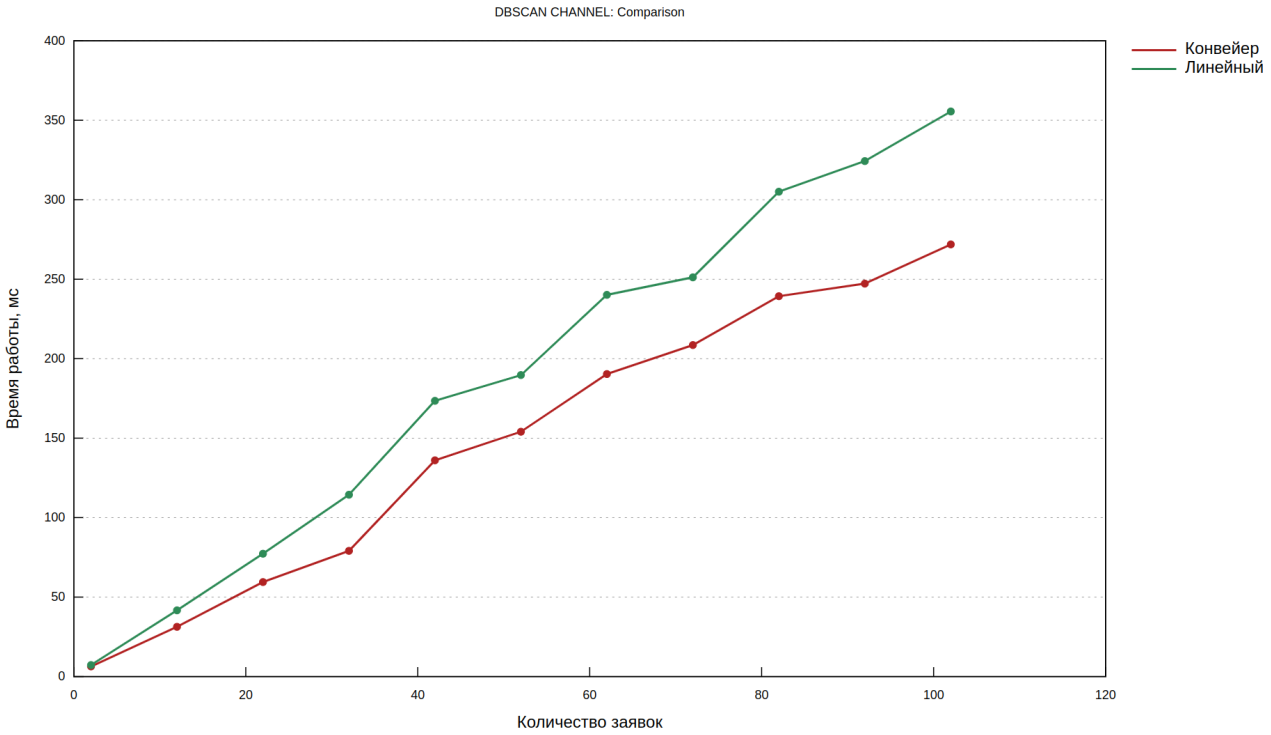

Рис. 4.1: Время выполнения параллельного и последовательного конвейеров в зависимости от длины очереди

4.2 Пример работы и анализ результата

Пример работы программы приведен на рисунке 4.2.

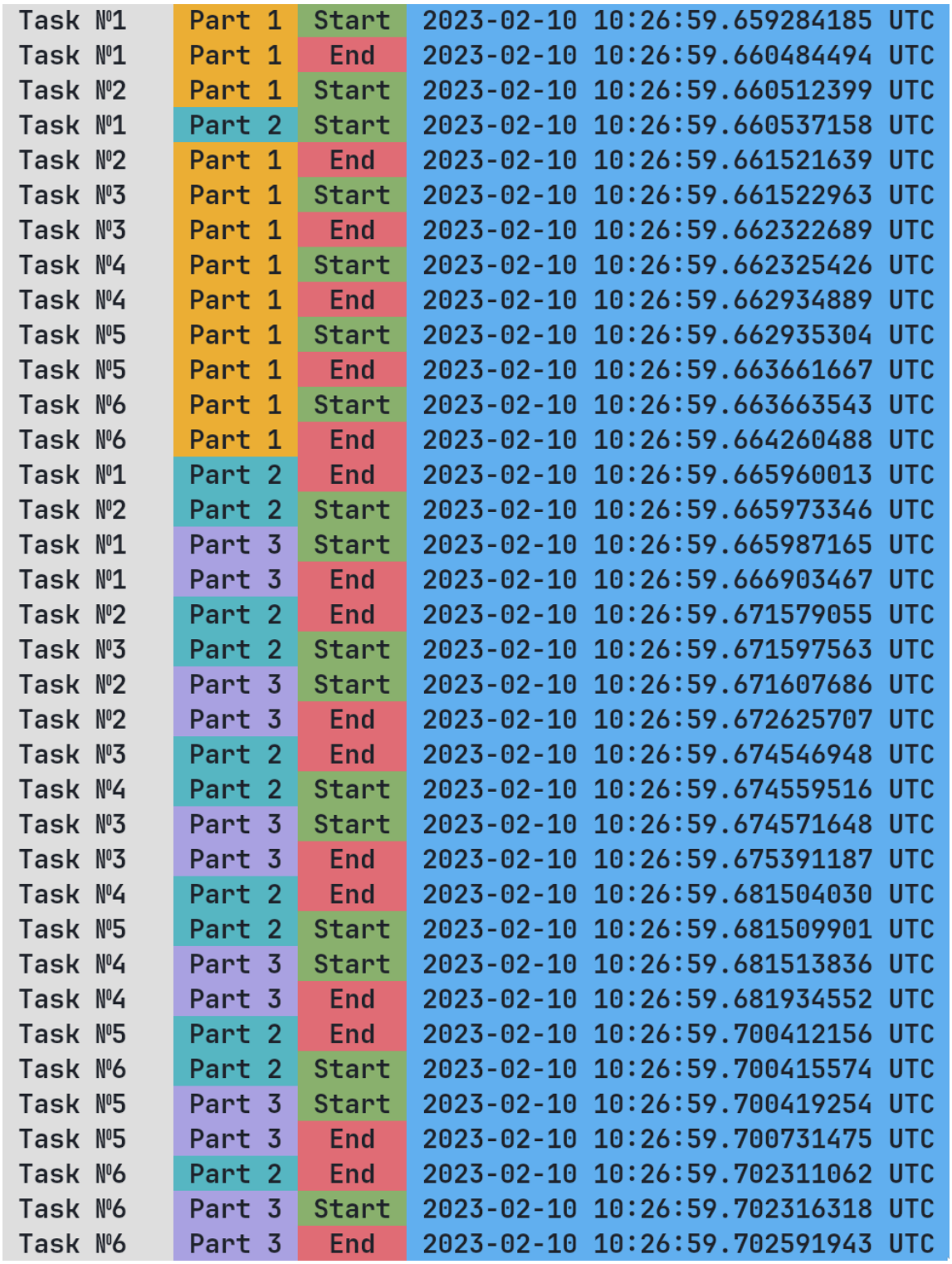

Рис. 4.2: Пример работы программы

Вывод

В данном разделе были сравнены алгоритмы по времени. Выявлено, что конвейерная обработка быстрее последовательной в среднем на 20%

Заключение

В ходе выполнения лабораторной работы были решены следующие задачи:

- описана организация конвейерной обработки данных;
- реализованы программы, реализующее конвейер с тремя лентами;
- исследована зависимость времени работы конвейера от количества потоков, на которых он работает

Благодаря конвейерной обработке данных возможна крайне удобная реализация задач, требующих поэтапной обработки некоторого набора данных. При этом схема обработки данных предоставляет простую схему параллельной обработки задач без конкуренции за данные, так как в определенный момент времени объект принадлежит только одной ленте конвейера.

Поставленная цель достигнута: описаны параллельные конвейерные вычисления.

Литература

[1] Параллельная обработка данных. Режим доступа: https://parallel.ru/vvv/lec1.html.

[2] 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.

[3] Wang Yiqiu, Gu Yan, Shun Julian. Theoretically-Efficient and Practical Parallel DBSCAN. Режим доступа: https://arxiv.org/abs/1912.06255. 2019.

[4] Rust [Электронный ресурс]. Режим доступа: https://www.rust-lang.org/. Дата обращения: 19.12.2022.

[5] Arch Linux [Электронный ресурс]. Режим доступа: https://archlinux.org/. Дата обращения: 19.12.2022.

[6] Процессор Intel® Core™ i5-11320H [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/217183/intel-core-i511320h-processor-8m-cache-up-to-4-50-ghz-with-ipu.html. Дата обращения: 19.12.2022.