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»

НАПРАВЛЕНИЕ ПОДГОТОВКИ «09.03.04 Программная инженерия»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

Название: Распараллеливание алгоритма DBSAN

Дисциплина: Анализ алгоритмов

Студент	ИУ7-56Б		Ковель А.Д.
	Группа	Подпись, дата	И. О. Фамилия
Преподаватель			Волкова Л.Л.
Преподаватель			Строганов Ю.В.
		Подпись, дата	И. О. Фамилия

Москва, 2022 г.

Оглавление

Введение

Цель лабораторной работы – описание параллельных конвейерных вычислений.

Задачи данной лабораторной:

- описать организацию конвейерной обработки данных;
- реализовать программ, реализующее конвейер с количеством лент не менее трех в однопоточной и многопоточной среде;
- исследовать зависимость времени работы конвейера от количества потоков, на которых он работает.

1 Аналитическая часть

В этом разделе представляется описание конвейерной обработки данных.

1.1 Конвейерные вычисления

Способ организации процесса в качестве вычислительного конвейера (pipeline) позволяет построить процесс, содержащий несколько независимых этапов [1], на нескольких потоках. Выигрыш во времени достигается при выполнении нескольких задач за счет параллельной работы ступеней, вовлекая на каждом такте новую задачу или команду. Для контроля стадии используются три основные метрики, описанные ниже.

1. Время процесса - это время, необходимое для выполнения одной стадии.
2. Время выполнения - это время, которое требуется с момента, когда работа была выполнена на предыдущем этапе, до выполнения на текущем.
3. Время простоя - это время, когда никакой работы не происходит и линии простаивают.

Для того, чтобы время простоя было минимальным, стадии обработки должны быть одинаковы по времени в пределах погрешности. При возникновении ситуации, в которой время процесса одной из линий больше, чем время других в N раз, эту линию стоит распараллелить на N потоков.

1.2 Алгоритм DBSCAN

Алгоритм DBSCAN [2] (Density Based Spatial Clustering of Applications with Noise), плотностный алгоритм для кластеризации пространственных данных с присутствием шума). Данный алгоритм является решением разбиения (изначально пространственных) данных на кластеры произвольной

формы [3]. Основная концепция алгоритма DBSCAN состоит в том, чтобы найти области высокой плотности, которые отделены друг от друга областями низкой насыщенности. Кучность региона измеряется в два шага. Для каждой точки считается количество соседей в окружности радиуса ϵ. Далее определяется область плотности, где для каждой точки в кластере окружности с радиусом ϵ содержит больше минимального количество точек.

Вывод

Плотностный алгоритм DBSCAN работает независимо от чтения и вывода из файла, что дает возможность реализовать конвейер.

2 Конструкторский часть

В данном разделе представлены схемы алгоритмов DBSCAN и конвейера.

2.1 Разработка алгоритмов

2.1.1 Разработка простого DBSCAN

На рисунке 2.1 приведена схема плотностного алгоритма DBSCAN.

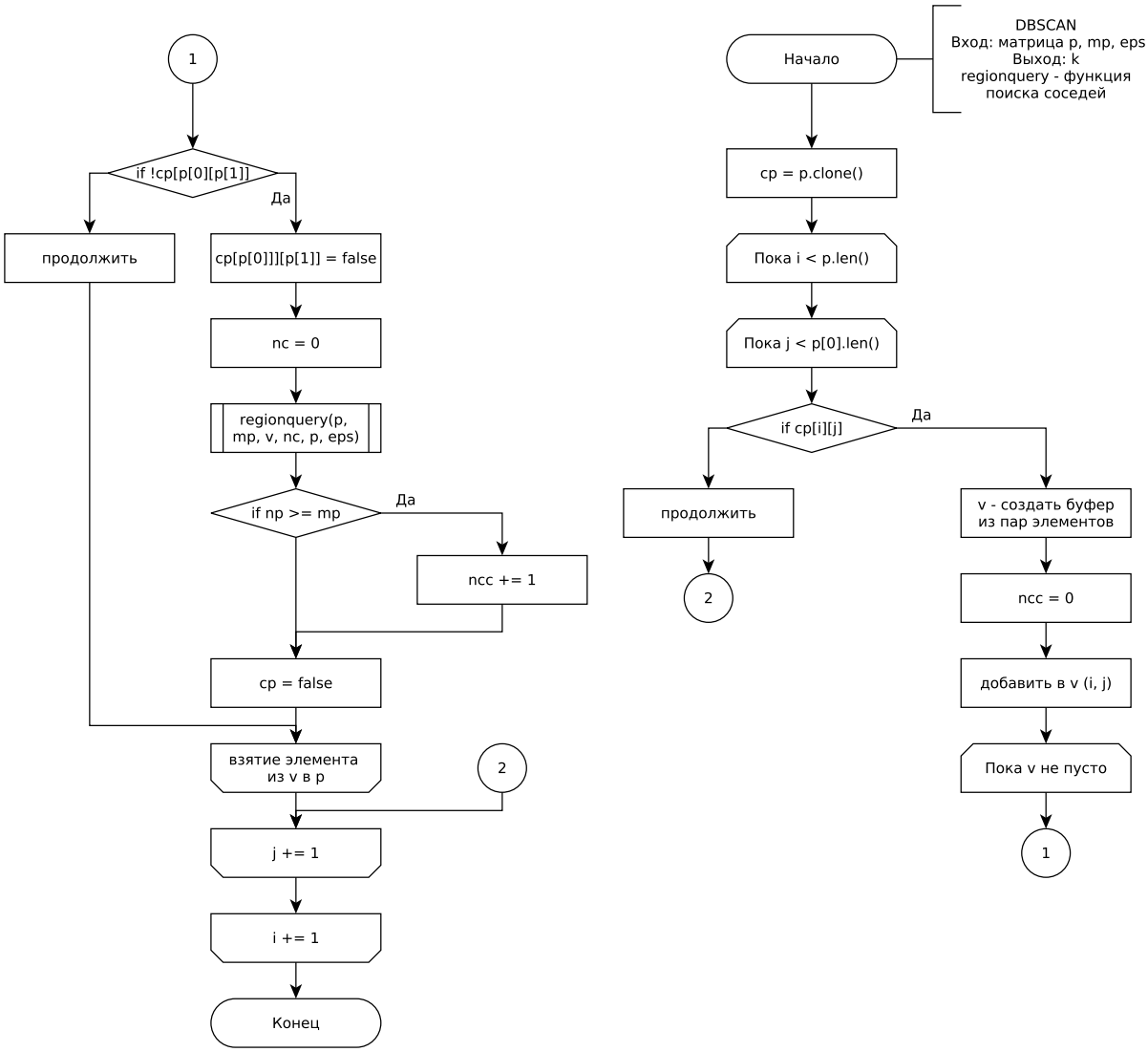

Рисунок 2.1 – Схема плотностного алгоритма DBSCAN

На рисунке 2.2 приведена схема функции поиска ближайшей соседний

точки в кластере.

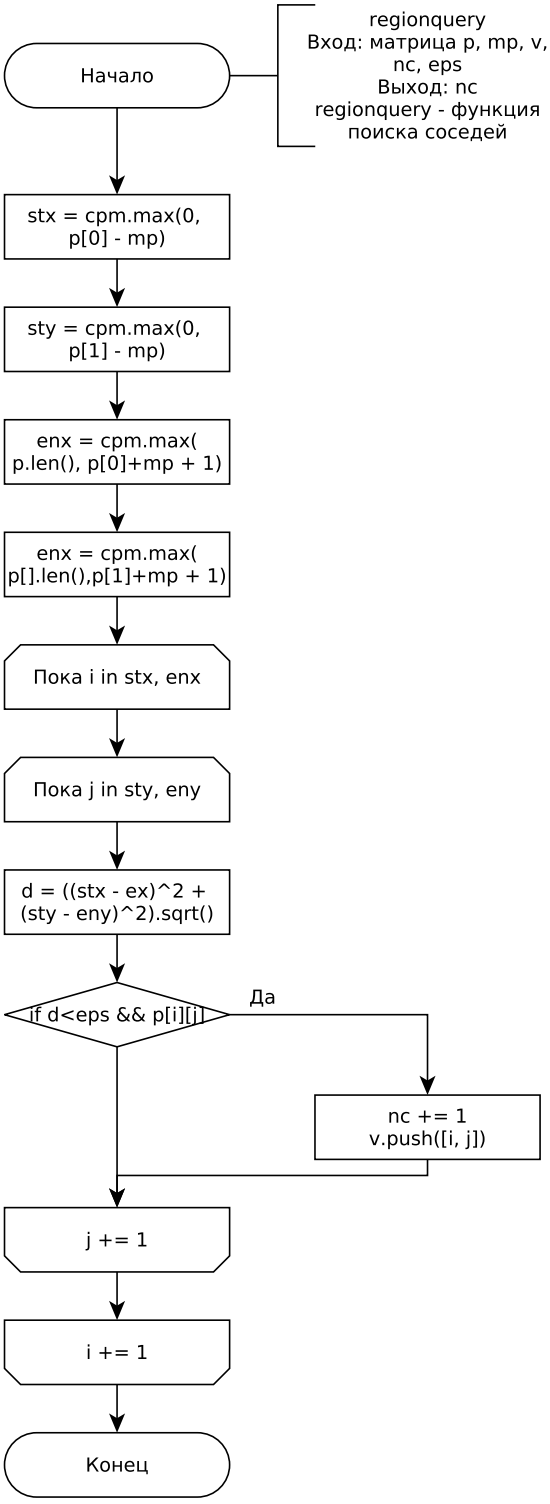

Рисунок 2.2 – Схема функции regionquery

2.1.2 Разработка конвейерных вычислений

На рисунке 2.3 приведена схема конвейерной линии.

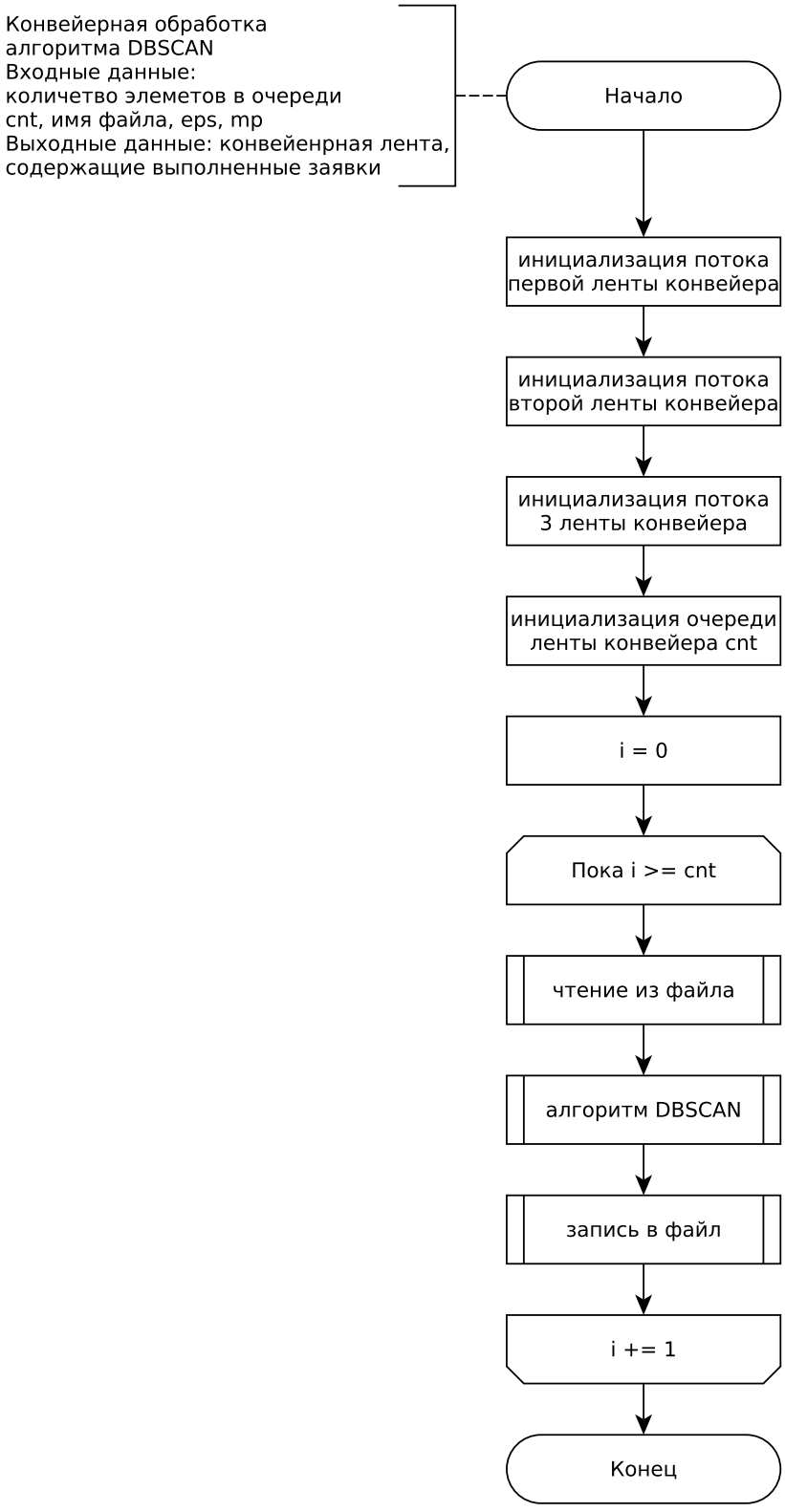

Рисунок 2.3 – Схема алгоритма параллельного DBSCAN

Вывод

Были разработаны схемы алгоритмов, необходимых для решения задачи.

3 Технологический часть

В данном разделе приведены требования к программному обеспечению, средства реализации и листинга кода.

3.1 Требования к программе

Программное обеспечение должно удовлетворять следующим требованиям:

- на вход подается имя файла;
- используется параллелизация программы;
- возможно измерение реального времени.

3.2 Средства реализации

Для реализации ПО был выбран язык программирования Golang [4]. В данном языке есть все требующиеся инструменты для данной лабораторной работы. В качестве среды разработки была выбрана среда VS Code [5], запуск происходил через команду go run main.go.

3.3 Реализации алгоритма

На листинге 3.1 приведена реализация простого алгоритма DBSCAN.

Листинг 3.1 – Реализация простого алгоритма DBSCAN

```
1  func DbscanAlgorithm(points structures.Matrix, minPtx int, eps
       float64) structures.Matrix {
2    clusterCount := 0
3    currentPoint := points
4    img := structures.Allocate(points.Rows, points.Cols)
5    color := 0
6    for i := 0; i < points.Rows; i++ {
7      for j:=0; j < points.Cols; j++ {
8        if currentPoint.Values[i][j] == 1 {
9          var v1 []int
10         var v2 []int
11         v1 = append(v1, i)
12         v2 = append(v2, j)
13         neighborCountCheck := 0
14         for k := 0; k < len(v1); k++ {
15           if currentPoint.Values[v1[k]][v2[k]] == 1 {
16             currentPoint.Values[v1[k]][v2[k]] = 0
17             neighborCount := 0
18             _, startX := MinMax([]int{0, v1[k] - minPtx})
19             _, startY := MinMax([]int{0, v2[k] - minPtx})
20             endX, _ := MinMax([]int{points.Rows, v1[k] + minPtx +
                   1})
21             endY, _ := MinMax([]int{points.Cols, v2[k] + minPtx +
                   1})
22             if neighborCount >= minPtx {
23               neighborCountCheck++
24               img.Values[v1[k]][v2[k]] = color
25             }
26           }
27         }
28         if neighborCountCheck > 0 {
29           clusterCount++
30           color++
31         }
32         currentPoint.Values[i][j] = 0
33       }
34     }
35   }
36   return img
37 }
```

На листинге 3.2 приведена реализация функции поиска ближайших точек regionquery.

Листинг 3.2 – Реализация функции regionquery

```
for x := startX; x < endX; x++ {
  for y := startY; y < endY; y++ {
    distance := math.Sqrt(math.Pow(float64(startX - endX), 2) +
       math.Pow(float64(startY - endY), 2))

    if distance <= eps && currentPoint.Values[x][y] == 1 {
      neighborCount++

      v1 = append(v1, x)
      v2 = append(v2, y)
    }
  }
}
```

На листинге 3.3 приведена реализация параллельного конвейера.

Листинг 3.3 – Реализация параллельного алгоритма DBSCAN

```
func Pipeline(count int, ch chan int, filename string, minPtx
   int, eps float64) *structures.Queue {
  first := make(chan *structures.PipeTask, count)
  second := make(chan *structures.PipeTask, count)
  third := make(chan *structures.PipeTask, count)
  line := InitQueue(count)
  get_file := func() {
    for {
      select {
        case pipe_task := <- first:
        pipe_task.Generated = true

        pipe_task.Start_generating = time.Now()

        pipe_task.Source = dbscan.ReadFile(filename)

        pipe_task.End_generatig = time.Now()

        second <- pipe_task
      }
    }
```

```
21  }
22  get_dbscan := func() {
23    for {
24      select {
25        case pipe_task := <- second:
26        pipe_task.Dbscan_mode = true
27
28        pipe_task.Start_dbscan = time.Now()
29
30        pipe_task.Dbscan =
             dbscan.DbscanAlgorithm(pipe_task.Source, minPtx, eps)
31
32        pipe_task.End_dbscan = time.Now()
33
34        third <- pipe_task
35      }
36    }
37  }
38  match := func ()  {
39    for {
40      select {
41        case pipe_task := <- third:
42        pipe_task.Pattern_matched = true
43
44        pipe_task.Start_match = time.Now()
45        pipe_task.Pattern_index =
             dbscan.SaveFile(pipe_task.Dbscan)
46        pipe_task.End_match = time.Now()
47
48        line.Enqueue(pipe_task)
49        if (pipe_task.Num == count - 1) {
50          ch <- 0
51        }
52      }
53    }
54  }
55  go get_file()
56  go get_dbscan()
57  go match()
58
59  for i := 0; i < count; i++ {
```

```
60      pipe_task := new(structures.PipeTask)
61      pipe_task.Num = i + 1
62      first <- pipe_task
63    }
64    return line
65  }
```

На листинге 3.4 приведена реализация синхронного конвейера.

Листинг 3.4 – Реализация функции параллелизации матрицы точек

```
1  func gen_string_sync(task *structures.PipeTask)
       *structures.PipeTask {
2    task.Generated = true
3    task.Start_generating = time.Now()
4    task.Source = dbscan.ReadFile("100.txt")
5    task.End_generatig = time.Now()
6    return task
7  }
8  func get_table_sync(task *structures.PipeTask)
       *structures.PipeTask {
9    task.Dbscan_mode = true
10   task.Start_dbscan = time.Now()
11   task.Dbscan = dbscan.DbscanAlgorithm(task.Source, 2, 2)
12   task.End_dbscan = time.Now()
13   return task
14 }
15 func match_sync(task *structures.PipeTask) *structures.PipeTask {
16   task.Pattern_matched = true
17   task.Start_match = time.Now()
18   task.Pattern_index = dbscan.SaveFile(task.Dbscan)
19   task.End_match = time.Now()
20   return task
21 }
22 func Sync(count int) *structures.Queue {
23   line_first := InitQueue(count)
24   line_second := InitQueue(count)
25   line_third := InitQueue(count)
26   for i := 0; i < count; i++ {
27     pipe_task := new(structures.PipeTask)
28     pipe_task = gen_string_sync(pipe_task)
29     line_first.Enqueue(pipe_task)
30     if (len(line_first.Queue) != 0) {
31       pipe_task = get_table_sync(line_first.Dequeue())
32       line_second.Enqueue(pipe_task)
33       if (len(line_second.Queue) != 0)
34         pipe_task = match_sync(line_second.Dequeue())
35         line_third.Enqueue(pipe_task)
36       }
37     }
```

```
38    }
39    return line_third
40 }
```

3.4 Тестовые данные

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы DBSCAN. Применена методология черного ящика. Тесты для всех алгоритмов пройдены *успешно*.

Таблица 3.1 – Функциональные тесты

Вход	Ожидаемый результат	Результат послед.	Результат паралл.
$()$	0	0	0
(0)	0	0	0
$(0 \quad 1)$	1	1	1
(1)	1	1	1
$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$	3	3	3
$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	2	2	2
$(1 \quad 0 \quad 0)$	1	1	1
$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix}$	4	4	4

Вывод

Написано и протестировано программное обеспечение для решения поставленной задачи.

4 Исследовательская часть

4.1 Технические характеристики

Тестирование выполнялось на устройстве со следующими техническими характеристиками:

- Операционная система Pop!_OS 22.04 LTS [7] Linux [8];
- Оперативная память 16 Гбайт;
- Процессор AMD® Ryzen 7 2700 eight-core processor × 16 [9].

Во время тестирования устройство было подключено к блоку питания и не нагружено никакими приложениями, кроме встроенных приложений и системой тестирования.

4.2 Демонстрация работы программы

Результат работы программы, в которой выводится время работы алгоритма представлено на рисунке 4.1.

```
(work) alexander@pop-os:~/Documents/bmstu/ics7-aa/lab-05/app$ go run main.go
Алгоритм DBSCAN
                          STARTING TIME
+-----------------------------------+
|  N|    Чтение|    DBSCAN|    Запись|
+-----------------------------------+
|  0|        0s|1.362863ms|39.346903ms|
|  1|1.357483ms|39.269226ms|71.552029ms|
|  2|2.592173ms|71.517373ms|97.895136ms|
|  3|4.406752ms|97.81763ms|124.739283ms|
|  4|5.400215ms|124.647268ms|150.251875ms|
|  5|6.671605ms|150.168658ms|175.661533ms|
|  6|7.656211ms|175.568707ms|204.62165ms|
|  7|8.964721ms|204.529095ms|229.531641ms|
|  8|10.040259ms|229.453402ms|255.624203ms|
|  9|11.316298ms|255.53332ms|280.916939ms|
| 10|13.144422ms|280.840294ms|306.643247ms|
| 11|14.44625ms|306.527818ms|332.481426ms|
| 12|16.040922ms|332.408278ms|364.662718ms|
| 13|17.607601ms|364.584269ms|399.569094ms|
| 14|18.913066ms|399.48208ms|427.691614ms|
| 15|20.613538ms|427.597205ms|453.138081ms|
| 16|22.319171ms|453.062498ms|479.098964ms|
| 17|24.610473ms|479.023961ms|505.296164ms|
| 18|26.620072ms|505.214278ms|531.911677ms|
+-----------------------------------+
                          FINISHING TIME
+-----------------------------------+
|  N|    Чтение|    DBSCAN|    Запись|
+-----------------------------------+
|  0|1.355419ms|39.264897ms|46.412454ms|
|  1|2.591722ms|71.510891ms|77.238457ms|
|  2|4.406091ms|97.813672ms|110.997829ms|
|  3|5.399895ms|124.64282ms|137.78603ms|
|  4|6.671134ms|150.163798ms|163.029513ms|
|  5| 7.65592ms|175.564519ms|189.175776ms|
|  6|8.963799ms|204.524335ms|219.234735ms|
|  7|10.039838ms|229.449084ms|244.670443ms|
|  8|11.315075ms|255.528491ms|269.153634ms|
|  9|13.143791ms|280.835835ms|296.944044ms|
| 10|14.445899ms|306.52381ms|321.704752ms|
| 11|16.039559ms|332.40412ms|345.419798ms|
| 12|17.60705ms|364.5796ms|379.236289ms|
| 13|18.912685ms|399.476509ms|414.693209ms|
| 14|20.613118ms|427.592776ms|441.909059ms|
| 15|22.318339ms|453.05831ms|467.998405ms|
| 16|24.609662ms|479.020244ms|495.013656ms|
| 17|26.619491ms|505.209079ms|520.386053ms|
| 18|28.439992ms|531.829562ms|547.161238ms|
+-----------------------------------+
                          Время простоя
+-----------------------------------+
|  N|                Время простоя|
+-----------------------------------+
|  1|                     12.745µs|
|  2|                      82.57µs|
|  3|                  250.283138ms|
+-----------------------------------+
```

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения реализации алгоритмов

Результаты замеров времени работы реализаций конвейерной линии приведены в таблице 4.1. Сравнение проводилось между простым алгоритмом и параллельного алгоритма при исполнении на 16 потоках.

Таблица 4.1 – Замеры времени работы на очереди размером 18

№	Начало обработки заявки					
	Параллельно, мс.			Синхронно, мс.		
	1	2	3	1	2	3
1	0	1.362	39.346	0	2.264	46.412
2	1.357	39.264	71.552	2.591	71.510	77.238
3	2.592	71.517	97.895	4.406	97.813	110.997
4	4.406	97.817	124.739	5.399	124.642	137.786
5	5.400	124.647	150.251	6.671	150.163	163.029
6	6.671	150.168	175.661	7.655	175.564	189.175
7	7.656	175.568	204.621	8.963	204.524	219.234
8	8.964	204.529	229.531	10.039	229.449	244.670
9	10.040	229.453	255.624	11.315	255.528	269.153
10	11.316	255.533	280.916	13.143	280.835	296.944
11	13.144	280.840	306.643	14.445	306.523	321.704
12	14.446	306.527	332.481	16.039	332.404	345.419
13	16.040	332.408	364.662	17.607	364.579	379.236
14	17.607	364.584	399.569	18.912	399.476	414.693
15	18.913	399.482	427.691	20.613	427.592	441.909
16	20.613	427.597	453.138	22.318	453.058	467.998
17	22.319	453.062	479.098	24.609	479.020	495.013
18	24.610	479.023	505.296	26.619	505.209	520.386

Из таблицы можно сделать вывод, что распараллеленный конвейер выполняет работу на 10% быстрее, чем синхронный.

Вывод

В данном разделе были сравнены алгоритмы по времени. Выявлено, что конвейерная обработка быстрее последовательной на 10% (2 мс. при

размере матрицы 100 на 100).

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.

В данном разделе были сравнены алгоритмы по времени. Выявлено, что конвейерная обработка быстрее последовательной на 10% (2 мс. при размере матрицы 100 на 100).

Поставленная цель достигнута: описаны параллельные конвейерные вычисления.

Список использованных источников

[1] Pipeline Processing in digital logic design [Электронный ресурс]. Режим доступа: https://www.fullchipdesign.com/pipeline_space_time_architecture.htm (дата обращения: 15.12.2022).

[2] Большакова Е.И. Клышинский Э.С. Ландэ Д.В. Носков А.А. Пескова О.В. Ягунова Е.В. Автоматическая обработка текстов на естественном языке и компьютерная лингвистика. – М.: МИЭМ, 2011. Т. 500. С. 197–200.

[3] С.А. Иванов. Мировая система научной коммуникации kar информационное пространство. – М.:, 2001., 2001. Т. 1. С. 1123–1126.

[4] Golang Документация[Электронный ресурс]. Режим доступа: https://go.dev/doc/ (дата обращения: 24.09.2022).

[5] Vscode Документация[Электронный ресурс]. Режим доступа: https://ru.bmstu.wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D0%BF%D0%BE%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C (дата обращения: 02.11.2022).

[6] Функция Instant::now() модуля std::time rust [Электронный ресурс]. Режим доступа: https://doc.rust-lang.org/std/time/index.html (дата обращения: 28.09.2022).

[7] Pop OS 22.04 LTS [Электронный ресурс]. Режим доступа: https://pop.system76.com (дата обращения: 04.09.2022).

[8] Linux – Документация [Электронный ресурс]. Режим доступа: https://docs.kernel.org (дата обращения: 24.09.2022).

[9] Процессор AMD® Ryzen 7 2700 eight-core processor × 16 [Электронный ресурс]. Режим доступа: https://www.amd.com/en/products/cpu/amd-ryzen-7-2700 (дата обращения: 04.09.2022).