



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 №6 по курсу "Анализ алгоритмов"

Тема Муравьиный алгоритм

Студент Зайцева А.А.

Группа ИУ7-52Б

Преподаватели Волкова Л.Л.

Москва — 2021 г.

# Оглавление

<b>Введение</b>	<b>3</b>
<b>1 Аналитическая часть</b>	<b>4</b>
1.1 Задача коммивояжера . . . . .	4
1.2 Алгоритм полного перебора для решения задачи коммивояжера . . . . .	4
1.3 Муравьиный алгоритм для решения задачи коммивояжера .	4
1.4 Вывод из аналитической части . . . . .	6
<b>2 Конструкторская часть</b>	<b>7</b>
2.1 Разработка конвейера . . . . .	7
2.2 Схемы этапов алгоритма стандартизации . . . . .	8
2.3 Схема линейного алгоритма стандартизации . . . . .	12
2.4 Схемы параллельной обработки данных конвейером для стандартизации . . . . .	14
2.5 Вывод из конструкторской части . . . . .	19
<b>3 Технологическая часть</b>	<b>20</b>
3.1 Выбор средств реализации . . . . .	20
3.2 Сбор статистики . . . . .	20
3.3 Реализация алгоритмов . . . . .	21
3.4 Тестирование . . . . .	25
3.5 Вывод из технологической части . . . . .	27
<b>4 Исследовательская часть</b>	<b>28</b>
4.1 Технические характеристики . . . . .	28
4.2 Сравнение времени выполнения реализаций алгоритмов . . .	28
4.3 Анализ статистики параллельного конвейера . . . . .	29
4.4 Вывод из исследовательской части . . . . .	31
<b>Заключение</b>	<b>32</b>
<b>Список использованной литературы</b>	<b>33</b>

# Введение

Муравьиные алгоритмы представляют собой новый перспективный метод решения задач оптимизации, в основе которого лежит моделирование поведения колонии муравьев. Колония представляет собой систему с очень простыми правилами автономного поведения особей. Однако, несмотря на примитивность поведения каждого отдельного муравья, поведение всей колонии оказывается достаточно разумным. Эти принципы проверены временем — удачная адаптация к окружающему миру на протяжении миллионов лет означает, что природа выработала очень удачный механизм поведения [1].

Целью данной работы является реализация муравьиного алгоритма для решения задачи коммивояжера и приобретение навыков параметризации алгоритмов.

В рамках выполнения работы необходимо решить следующие задачи:

- 1) реализовать алгоритм полного перебора для решения задачи коммивояжера;
- 2) изучить и реализовать муравьиный алгоритм для решения задачи коммивояжера;
- 3) провести параметризацию муравьиного алгоритма на трех классах данных и подобрать оптимальные параметры;
- 4) провести сравнительный анализ трудоемкостей реализаций.

# 1 Аналитическая часть

В данном разделе будет приведена теория, необходимая для разработки и реализации двух алгоритмов решения задачи коммивояжера: алгоритма полного перебора и муравьиного алгоритма.

## 1.1 Задача коммивояжера

В задаче коммивояжера рассматривается  $n$  городов и матрица попарных расстояний между ними. Требуется найти такой порядок посещения городов, чтобы суммарное пройденное расстояние было минимальным, каждый город посещался ровно один раз и коммивояжер вернулся в тот город, с которого начал свой маршрут. Другими словами, во взвешенном полном графе требуется найти гамильтонов цикл минимального веса [2].

## 1.2 Алгоритм полного перебора для решения задачи коммивояжера

Суть алгоритма полного перебора для решения задачи коммивояжера заключается в переборе всех вариантов путей и нахождении кратчайшего. Преимуществом является его результат - точное решение, недостатком - длительность вычислений при относительно небольшом пространстве поиска [3]. Эта задача является NP-трудной, и точный переборный алгоритм ее решения имеет факториальную сложность [1].

## 1.3 Муравьиный алгоритм для решения задачи коммивояжера

Моделирование поведения муравьев связано с распределением феромона на тропе — ребре графа в задаче коммивояжера. При этом вероятность

включения ребра в маршрут отдельного муравья пропорциональна количеству феромона на этом ребре, а количество откладываемого феромона пропорционально длине маршрута. Чем короче маршрут, тем больше феромона будет отложено на его ребрах. При этом избежать преждевременной сходимости можно, моделируя отрицательную обратную связь в виде испарения феромона. С учетом особенностей задачи коммивояжера, мы можем описать локальные правила поведения муравьев при выборе пути.

- Муравьи имеют собственную «память» в виде списка уже посещенных городов. Обозначим через  $J_{i,k}$  список городов, которые необходимо посетить муравью  $k$ , находящемуся в городе  $i$ .
- Муравьи обладают «зрением» — видимость есть эвристическое желание посетить город  $j$ , если муравей находится в городе  $i$ . Будем считать, что видимость обратно пропорциональна расстоянию между городами  $i$  и  $j$  —  $D_{ij}$ :  $\eta_{ij} = \frac{1}{D_{ij}}$ .
- Муравьи обладают «обонянием» — они могут улавливать след феромона, подтверждающий желание посетить город  $j$  из города  $i$ , на основании опыта других муравьев. Количество феромона на ребре  $(i, j)$  в момент времени  $t$  обозначим через  $\tau_{ij}(t)$ .

На этом основании мы можем сформулировать вероятностно-пропорционально правилу 1.1, определяющее вероятность перехода  $k$ -ого муравья из города  $i$  в город  $j$ :

$$P_{ij,k}(t) = \begin{cases} \frac{(\tau_{ij}(t))^\alpha (\eta_{ij}(t))^\beta}{\sum_{l \in J_{i,k}} (\tau_{il}(t))^\alpha (\eta_{il}(t))^\beta}, & j \in J_{i,k}, \\ 0, & \text{иначе} \end{cases}, \quad (1.1)$$

где  $\alpha$  — параметр влияния длины пути, при  $\alpha = 0$  алгоритм вырождается до жадного алгоритма (будет выбран ближайший город),  $\beta$  — параметр влияния феромона.

Пройдя ребро  $(i, j)$ , муравей откладывает на нем некоторое количество феромона, которое должно быть связано с оптимальностью сделанного выбора. Пусть  $T_k(t)$  есть маршрут, пройденный муравьем  $k$  к моменту времени  $t$ , а  $L_k(t)$  — длина этого маршрута. Пусть также  $Q$  — параметр,

имеющий значение порядка длины оптимального пути. Тогда откладываемое количество феромона может быть задано в виде 1.2:

$$\Delta\tau_{ij,k}(t) = \begin{cases} \frac{Q}{L_k(t)}, (i, j) \in T_k(t), \\ 0, \text{ иначе.} \end{cases} \quad (1.2)$$

Правила внешней среды определяют, в первую очередь, испарение феромона. Пусть  $p \in [0, 1]$  есть коэффициент испарения, тогда правило испарения имеет вид 1.3:

$$\tau_{ij}(t+1) = (1-p)\tau_{ij}(t) + \Delta\tau_{ij}(t), \Delta\tau_{ij} = \sum_{k=1}^m \Delta\tau_{ij,k}(t), \quad (1.3)$$

где  $m$  — количество муравьев в колонии.

В начале алгоритма количество феромона на ребрах принимается равным небольшому положительному числу. При этом необходимо следить, чтобы количество феромона на существующем ребре не обнулилось в ходе испарения. Общее количество муравьев остается постоянным и равным количеству городов, каждый муравей начинает маршрут из своего города.

## 1.4 Вывод из аналитической части

Были рассмотрены идеи и материалы, необходимые для разработки и реализации двух алгоритмов решения задачи коммивояжера: алгоритма полного перебора и муравьиного алгоритма.

## 2 Конструкторская часть

В данном разделе будет описана структура и принцип работы разрабатываемого конвейера, а также будут приведены схемы для этапов алгоритма стандартизации данных, для линейного алгоритма стандартизации, для главного и рабочих потоков параллельной реализации конвейера.

### 2.1 Разработка конвейера

Алгоритм стандартизации массива можно разделить на 3 этапа:

- 1) вычисление среднего значения;
- 2) вычисление стандартного отклонения;
- 3) вычисление стандартизованных значений.

Таким образом, конвейер состоит из 3 лент, каждая из которых выполняет соответствующий этап. Для каждой ленты в главном потоке создается отдельный поток.

Рабочий поток выполняется, пока не завершит обработку всех заявок, для чего в него передается общее количество задач, а в нем самом заводится счетчик уже обработанных заявок.

При этом в программе предусмотрен пул обработанных задач и 3 очереди заявок - по одной на каждую ленту. Очередь первой ленты заранее заполняется генератором заявок. Во вторую очередь заявки заносятся первой лентой после выполнения ею назначенной задачи, в третью очередь - второй лентой, в пул обработанных задач - третьей лентой.

Хотя для каждой ленты создана своя очередь, ко 2 и 3 очередям могут одновременно обратиться сразу два потока: предыдущий для записи в нее новой заявки и текущий (соответствующий номеру очереди) для получения новой заявки (в 1 очереди такая ситуация невозможна, так как она заполняется генератором заранее). Поэтому при доступе к элементам 2 и 3 очередей необходимо блокировать доступ для других потоков, для чего используются мьютексы, по одному для каждой очереди.

Для сбора статистики процесса обработки заявок конвейером предусмотрено сохранение информации о времени поступления заявки в очередную очередь и времени выхода из нее.

## **2.2 Схемы этапов алгоритма стандартизации**

На рисунках 2.1 - 2.3 приведены схемы этапов алгоритма стандартизации.



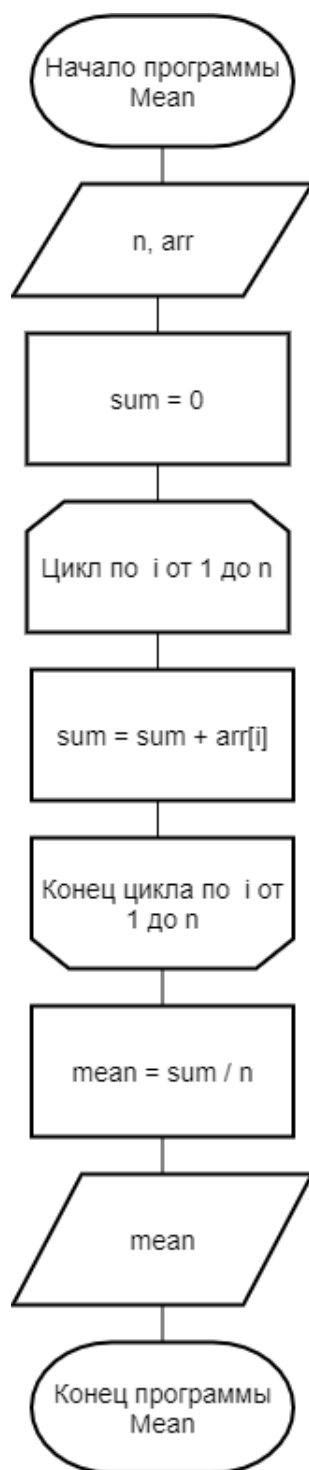


Рисунок 2.1 – Схема этапа поиска среднего значения в массиве

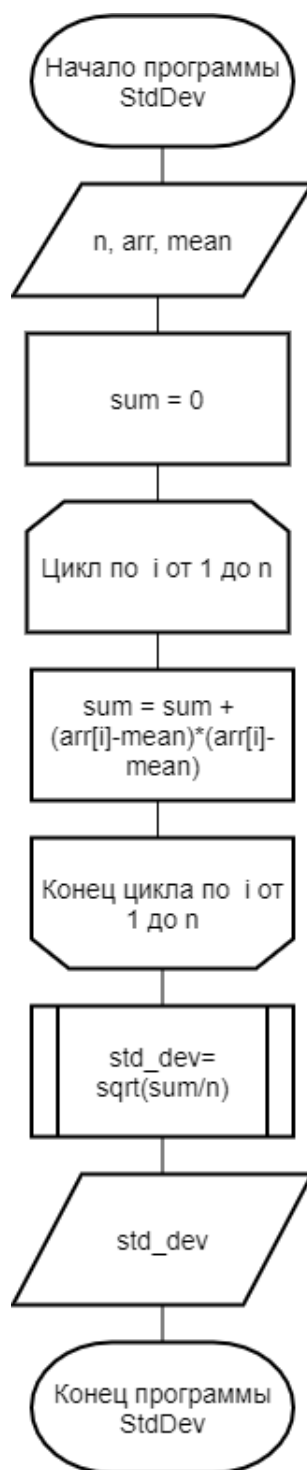


Рисунок 2.2 – Схема этапа поиска стандартного отклонения

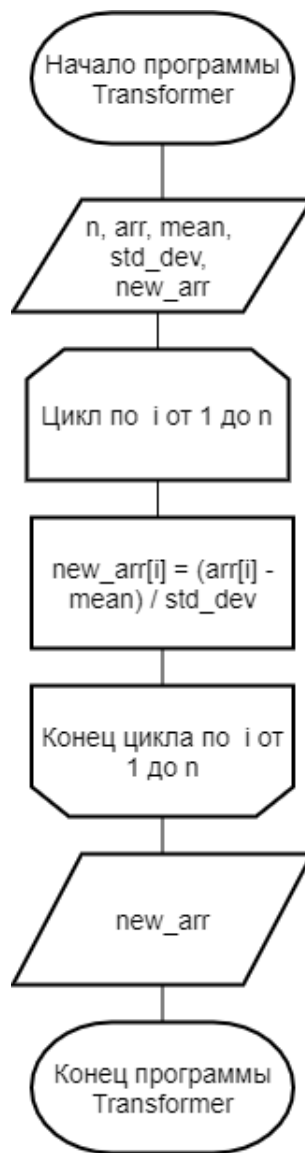


Рисунок 2.3 – Схема этапа преобразования (стандартизации) массива

## 2.3 Схема линейного алгоритма стандартизации

На рисунке 2.4 приведена схема линейного алгоритма обработки заявок на стандартизацию массивов.



Рисунок 2.4 – Схема линейного алгоритма обработки заявок на стандартизацию массивов

## 2.4 Схемы параллельной обработки данных конвейером для стандартизации

На рисунке 2.5 приведена схема главного потока параллельного конвейера для обработки заявок на стандартизацию массивов, который запускает и контролирует рабочие потоки.

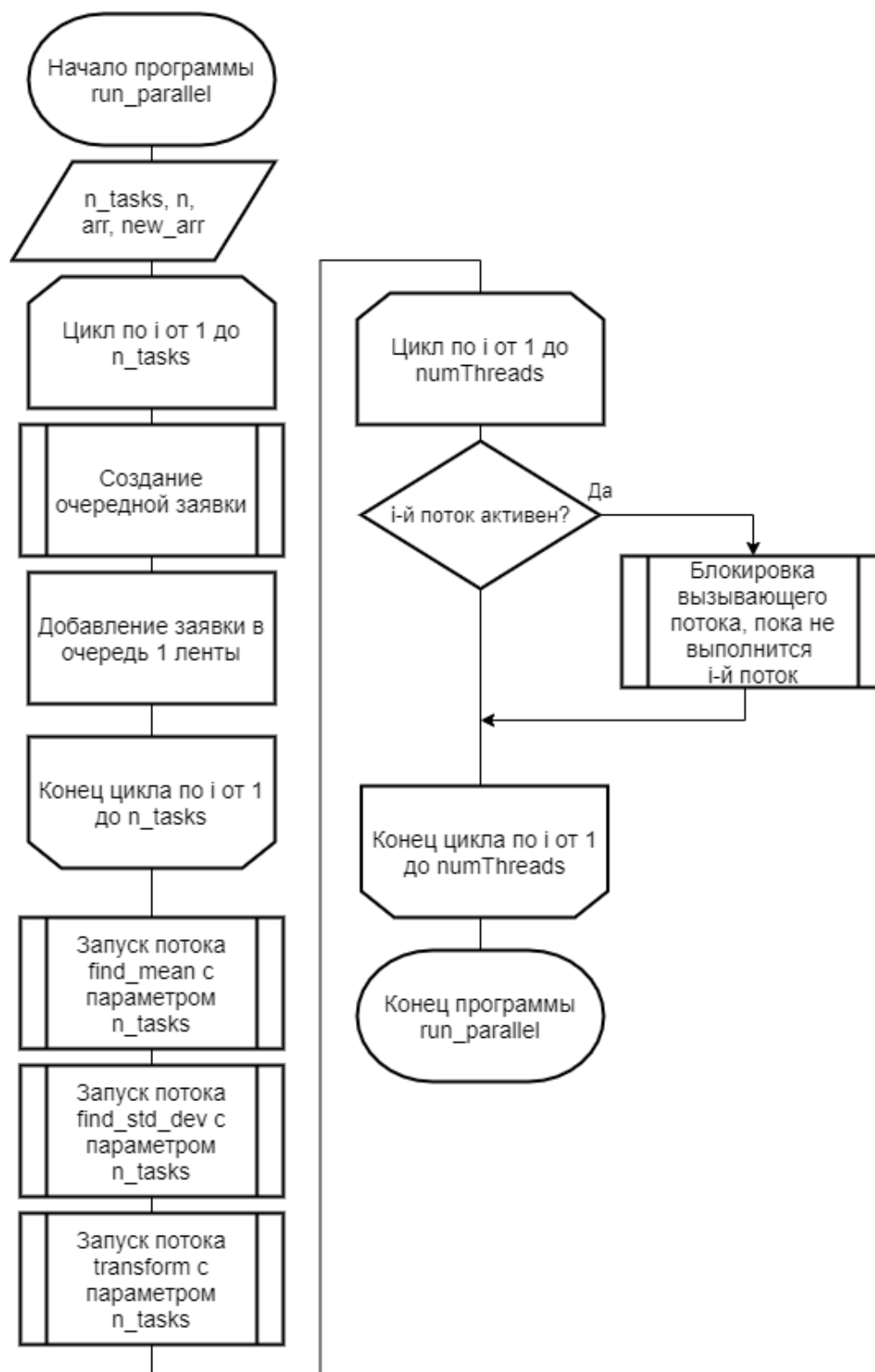


Рисунок 2.5 – Схема главного потока параллельного конвейера

На рисунках 2.6-2.8 приведены схемы рабочих потоков конвейера.

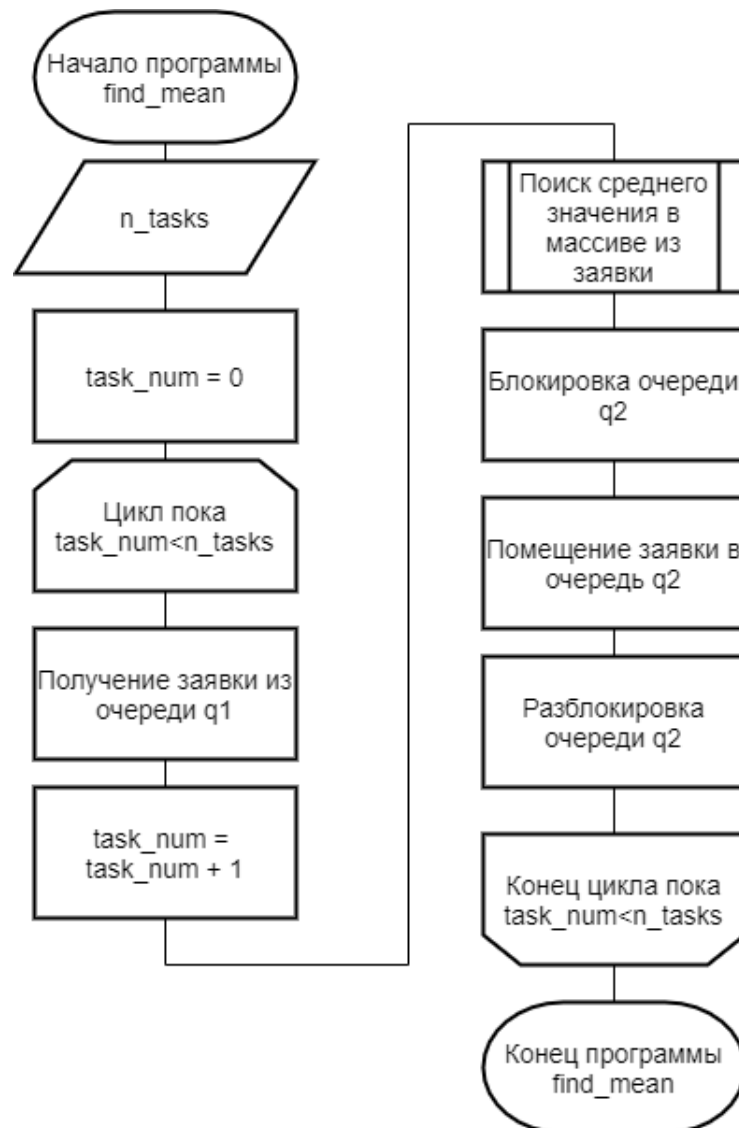


Рисунок 2.6 – Схема потока для поиска среднего значения в массиве



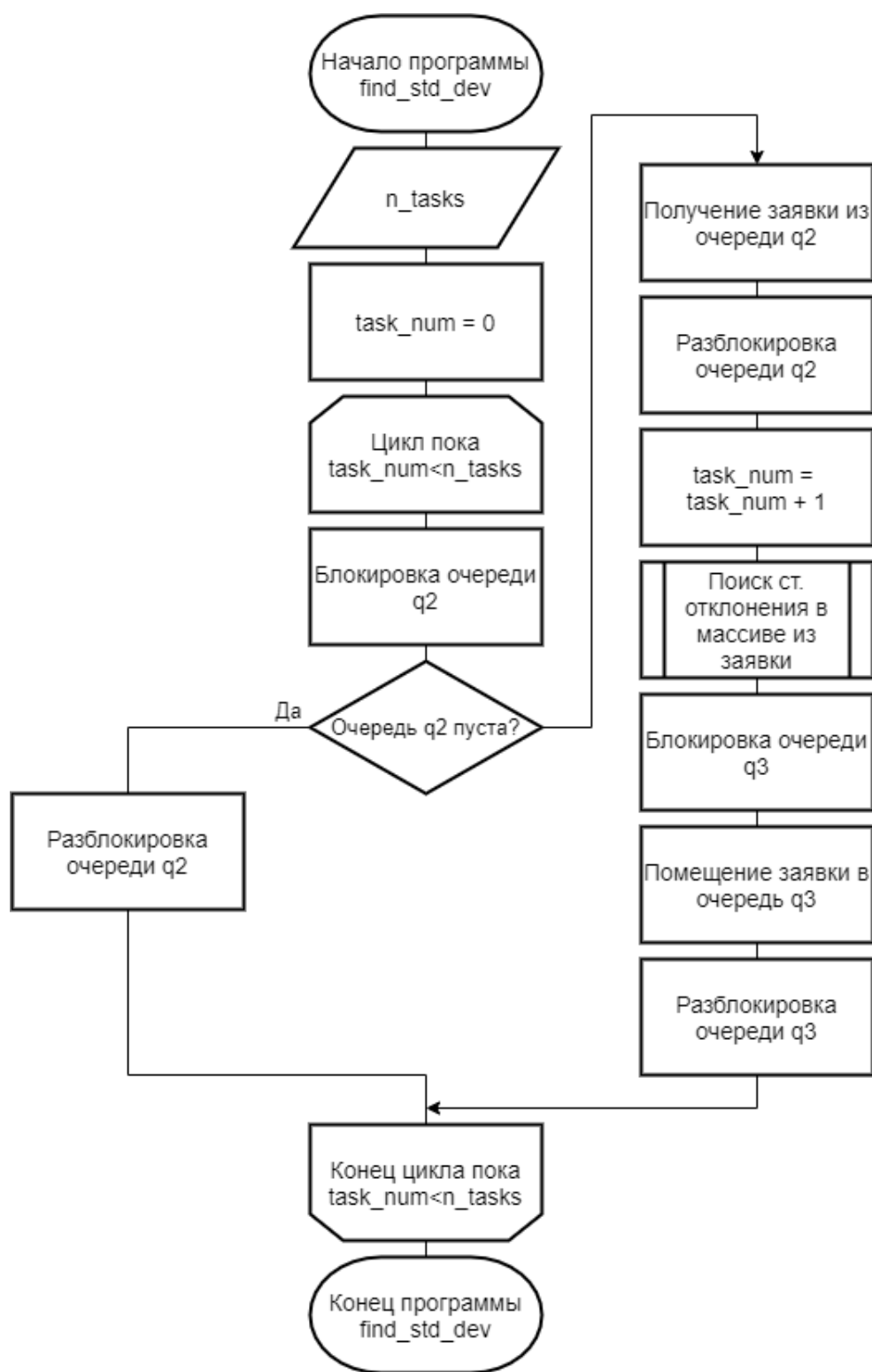


Рисунок 2.7 – Схема потока для поиска стандартного отклонения

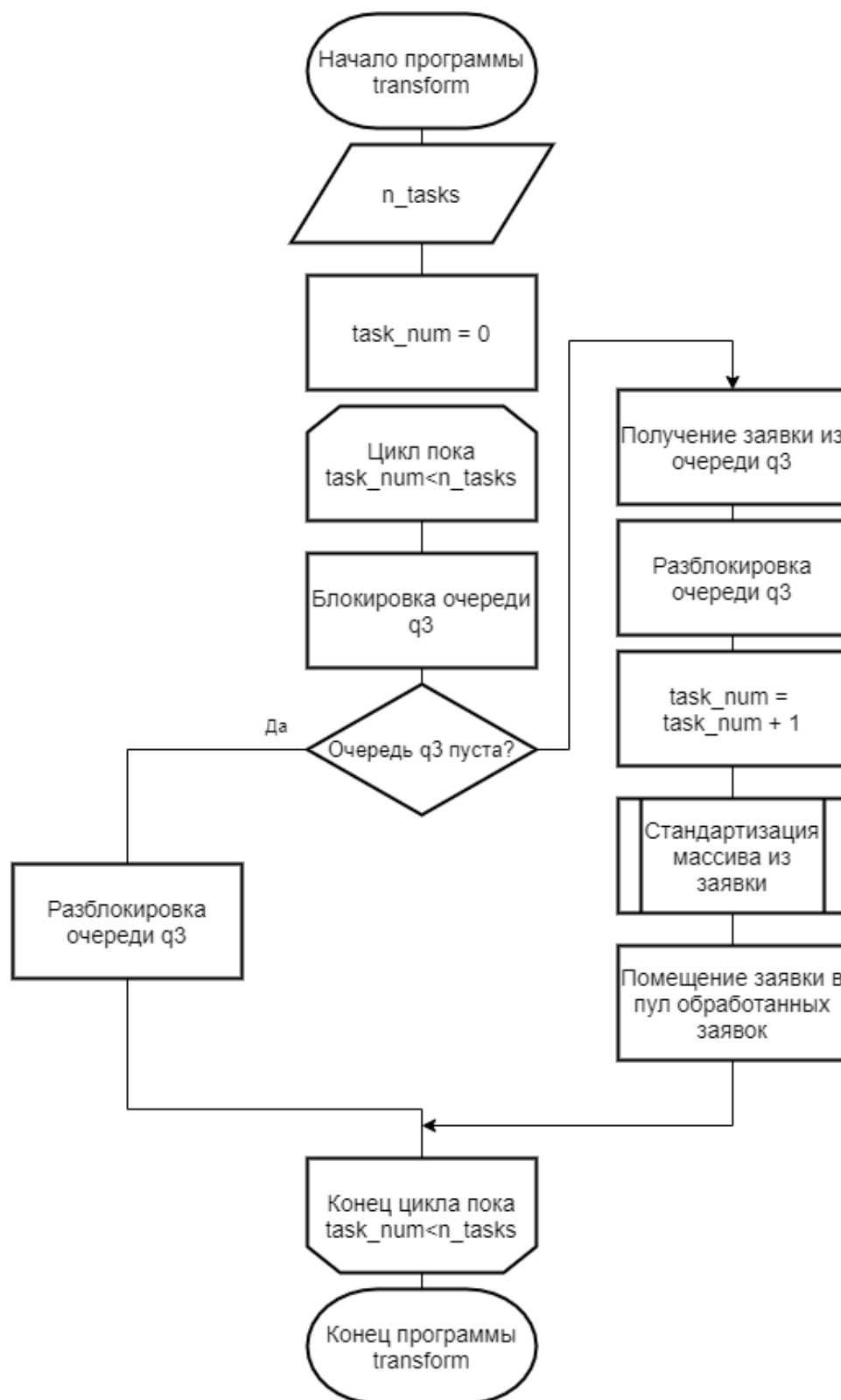


Рисунок 2.8 – Схема потока для преобразования (стандартизации) массива

## 2.5 Вывод из конструкторской части

Была описана структура и принцип работы разрабатываемого конвейера, а также приведены схемы разрабатываемых алгоритмов.

## 3 Технологическая часть

В данном разделе производится выбор средств реализации, а также приводятся: описание подхода к сбору статистики, листинги реализованных алгоритмов, результаты тестирования программы.

### 3.1 Выбор средств реализации

Основное требование к языку программирования в данной лабораторной работе - наличие в нем нативных потоков. Язык C++ обладает этим свойством [5] и уже использовался мною ранее, поэтому и был выбран.

Для замеров времени используется предоставляемый класс `system_clock::time_point` использующий данные системных часов в реальном времени [6], а для организации распараллеливания - `std::thread`.

В качестве среды разработки выбран “QT Creator” так как он был часто использована мною ранее.

### 3.2 Сбор статистики

Чтобы наглядно показать, что ленты конвейера работают параллельно, в класс завок добавлены отметки времени поступления заявки в очередную очередь (или в пул обработанных задач) и времени покидания очередной очереди. В процессе обработки заявок конвейером в лог выводится информация о каждом подобном событии для каждой заявки.

На выходе, когда все заявки обработаны системой, собирается следующая статистика.

- 1) На сколько время обработки N заявок снижено в параллельной реализации конвейера по сравнению с последовательной обработкой одним потоком. Для этого перед запуском и после завершения каждой из реализаций запоминаются данные системных часов в реальном времени, затем второе время вычитается из первого, и получается время работы этой реализации. Далее показания реализаций сравниваются.

- 2) Минимальное, максимальное и среднее время: проведенное заявкой в очереди, проведенное заявкой в системе. Для этого после обработки всех заявок для каждой из них вычисляется время, проведенной в каждой очереди, а также суммарные: время, проведенное в очередях и время, проведенное в системе. Затем вычисляется среднее, максимальное и минимальное каждого из этих значений среди всех заявок.

### 3.3 Реализация алгоритмов

В листинге 3.1 представлены реализации рабочих потоков параллельного конвейера.

Листинг 3.1 – Реализация рабочих потоков параллельного конвейера

```
1 void Conveyor::find_mean(size_t n_tasks)
2 {
3     size_t task_num = 0;
4
5     while (task_num < n_tasks)
6     {
7         std::shared_ptr<Standardizer> task = q1.front();
8         q1.pop();
9         task->out1 = system_clock::now();
10
11         task->find_mean(++task_num);
12
13         m2.lock();
14         q2.push(task);
15         task->in2 = system_clock::now();
16         m2.unlock();
17     }
18 }
19 void Conveyor::find_std_dev(size_t n_tasks)
20 {
21     size_t task_num = 0;
22
23     while(task_num < n_tasks)
24     {
25         m2.lock();
26         if (!this->q2.empty())
```

```

27     {
28         std::shared_ptr<Standardizer> task = q2.front();
29         q2.pop();
30         task->out2 = system_clock::now();
31         m2.unlock();
32
33         task->find_std_dev(++task_num);
34
35         m3.lock();
36         q3.push(task);
37         task->in3 = system_clock::now();
38         m3.unlock();
39     }
40     else
41         m2.unlock();
42 }
43 }
44 void Conveyor::transform(size_t n_tasks)
45 {
46     size_t task_num = 0;
47
48     while (task_num < n_tasks)
49     {
50         m3.lock();
51         if (!this->q3.empty())
52         {
53             std::shared_ptr<Standardizer> task = q3.front();
54             q3.pop();
55             task->out3 = system_clock::now();
56             m3.unlock();
57
58             task->transform(++task_num);
59
60             tasks.push_back(task);
61             task->out_system = system_clock::now();
62
63         }
64         else
65             m3.unlock();
66     }
67 }

```

В листинге 3.2 представлена реализация главного потока параллельного конвейера.

Листинг 3.2 – Реализация главного потока параллельного конвейера

```
1 void Conveyor::run_parallel(size_t n_tasks, size_t n, double *arr,
2     double *new_arr)
3 {
4     for (size_t i = 0; i < n_tasks; i++)
5     {
6         std::shared_ptr<Standardizer> new_task(new Standardizer(n, arr
7             , new_arr));
8         q1.push(new_task);
9         new_task->in1 = system_clock::now();
10    }
11
12    this->threads[0] = std::thread(&Conveyor::find_mean, this,
13        n_tasks);
14    this->threads[1] = std::thread(&Conveyor::find_std_dev, this,
15        n_tasks);
16    this->threads[2] = std::thread(&Conveyor::transform, this,
17        n_tasks);
18
19    for (size_t i = 0; i < numThreads; i++)
20    {
21        if (this->threads[i].joinable())
22            this->threads[i].join();
23    }
```

В листинге 3.3 представлена реализация последовательной обработки данных одним потоком.

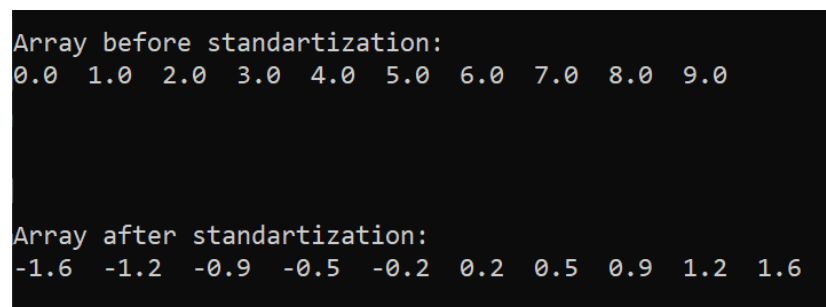
Листинг 3.3 – Реализация последовательной обработки данных одним  
ПОТОКОМ

```
1 void Conveyor::run_linear(size_t n_tasks, size_t n, double *arr,
2     double *new_arr)
3 {
4     for (size_t i = 0; i < n_tasks; i++)
5     {
6         std::shared_ptr<Standardizer> task(new Standardizer(n, arr,
7             new_arr));
8
9         task->find_mean(i + 1);
10        task->find_std_dev(i + 1);
11        task->transform(i + 1);
12
13        tasks.push_back(task);
14    }
15 }
```



## 3.4 Тестирование

Произведено тестирования алгоритма стандартизации программы по методу черного ящика: проведен запуск программного обеспечения для стандартизации массива из 10 элементов. Содержание массива до и после стандартизации приведено на рисунке 3.1.



```
Array before standartization:
0.0  1.0  2.0  3.0  4.0  5.0  6.0  7.0  8.0  9.0

Array after standartization:
-1.6 -1.2 -0.9 -0.5 -0.2 0.2 0.5 0.9 1.2 1.6
```

Рисунок 3.1 – Результат тестирования алгоритма стандартизации

Как видно на рисунке выше, цель стандартизации достигнута: получен массив с нулевым средним и стандартным отклонением, равным 1.

На рисунке 3.2 приведен пример вывода лога при обработке параллельным конвейером 5 заявок на стандартизацию массива из 100000 элементов.

```

Number of tasks: 5
T 1| P1 | Start| 20:31:04. 68
T 1| P1 | End  | 20:31:04. 76
T 2| P1 | Start| 20:31:04. 81
T 2| P1 | End  | 20:31:04. 87
T 3| P1 | Start| 20:31:04.107
T 3| P1 | End  | 20:31:04.114
T 4| P1 | Start| 20:31:04.151
T 4| P1 | End  | 20:31:04.324
T 5| P1 | Start| 20:31:04.355
T 5| P1 | End  | 20:31:04.362
T 1| P2 | Start| 20:31:04. 81
T 1| P2 | End  | 20:31:04.101
T 2| P2 | Start| 20:31:04.128
T 2| P2 | End  | 20:31:04.144
T 3| P2 | Start| 20:31:04.190
T 3| P2 | End  | 20:31:04.205
T 4| P2 | Start| 20:31:04.355
T 4| P2 | End  | 20:31:04.377
T 5| P2 | Start| 20:31:04.399
T 5| P2 | End  | 20:31:04.415
T 1| P3 | Start| 20:31:04.128
T 1| P3 | End  | 20:31:04.175
T 2| P3 | Start| 20:31:04.230
T 2| P3 | End  | 20:31:04.254
T 3| P3 | Start| 20:31:04.294
T 3| P3 | End  | 20:31:04.317
T 4| P3 | Start| 20:31:04.399
T 4| P3 | End  | 20:31:04.445
T 5| P3 | Start| 20:31:04.497
T 5| P3 | End  | 20:31:04.521

```

Рисунок 3.2 – Результат логирования

Как видно на рисунке выше, заявки действительно обрабатываются параллельно. Например, после завершения обработки первой заявки первым потоком, и ее помещения в очередь второго потока, параллельно начинается обработка второй заявки первым потоком и первой заявки вторым потоком.

На рисунке 3.3 приведен пример вывода собранной статистики при обработке параллельным конвейером 1000 заявок на стандартизацию массива из 100000 элементов.

```

##### STATISTICS (in ms) #####
Time in q1:      min=      1, max=   49726, mean=    25023
Time in q2:      min=      0, max=   11022, mean=     3774
Time in q3:      min=      0, max=   47449, mean=    39298
Total time in queue: min=      1, max=   93975, mean=    68096
Total time in system: min=    131, max=   94094, mean=    68302
##### ENS OF STATISTICS #####

```

Рисунок 3.3 – Пример собранной статистики

На рисунке 3.4 приведен пример вывода результатов сравнения времени выполнения при параллельной и линейной обработке 100 заявок на стандартизацию массива из 100000 элементов.

```
Duration parallel 00:00:10.477 (10477 ms total)
Duration linear   00:00:16.796 (16796 ms total)
Linear realization is 6319 ms (1.603 times) slower than parallel
```

Рисунок 3.4 – Результат сравнения времени выполнения параллельной и линейной реализаций

Более подробно статистика и сравнение времени работы реализаций будут рассмотрены в следующем разделе.

### 3.5 Вывод из технологической части

Был произведен выбор средств реализации, реализованы и протестированы последовательный и параллельный конвейерный подходы к стандартизации массива.

## 4 Исследовательская часть

### 4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- операционная система: Windows 10;
- оперативная память: 16 Гб;
- процессор: Intel® Core™ i5-8259U;
- количество ядер: 4;
- количество логических процессоров: 8.

Во время тестирования ноутбук был включен в сеть питания и нагружен только встроенными приложениями окружения и системой тестирования.

### 4.2 Сравнение времени выполнения реализаций алгоритмов

Сравнивалось время работы (обычное, по таймеру) последовательной стандартизации данных и стандартизации с использованием параллельного конвейера. Эти реализации сравнивались по времени обработки заявок на стандартизацию массива вещественных чисел из 10000 элементов в зависимости от количества заявок: 1, 5, 25 и от 50 до 250 с шагом 50.

Так как некоторые задачи выполняются достаточно быстро, а замеры времени имеют некоторую погрешность, они для каждой реализации и каждого количества заявок выполнялись 10 раз, а затем вычислялось среднее время работы.

На рисунке 4.1 приведены результаты сравнения времени выполнения реализаций алгоритмов.

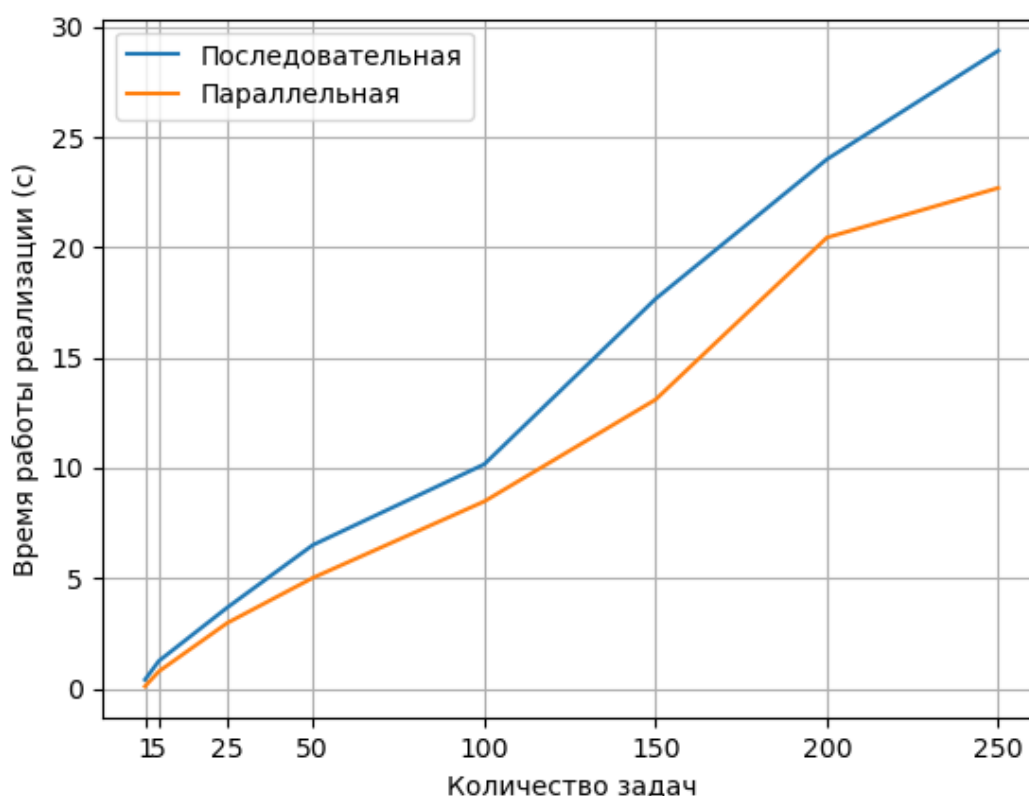


Рисунок 4.1 – Сравнение времени работы реализаций в зависимости от количества заявок

Как и ожидалось, параллельная реализация выполняется за меньшее количество времени в сравнении с линейной за счет того, что в ней одновременно на разных лентах (потоках) обрабатываются несколько заявок. Причем с ростом числа заявок разрыв между реализациями увеличивается.

### 4.3 Анализ статистики параллельного конвейера

На рисунке 4.2 приведен результат сбора статистики при обработке 1000 заявок на обработку массива из 100000 элементов.

```
##### STATISTICS (in ms) #####
Time in q1:      min=      1, max=    49726, mean=    25023
Time in q2:      min=      0, max=    11022, mean=     3774
Time in q3:      min=      0, max=    47449, mean=    39298
Total time in queue: min=      1, max=    93975, mean=    68096
Total time in system: min=    131, max=    94094, mean=    68302
##### ENS OF STATISTICS #####
```

Рисунок 4.2 – Результат сбора статистики

Как видно из рисунка, очереди с наибольшим максимальным временем нахождения в них заявки – первая и третья. Для первой очереди такой результат объясняется тем, что она заполняется генератором заранее, и последняя заявка находится в ней до того момента, пока все предшествующие ей не будут обработаны первой лентой. Это подтверждает и среднее время, проведенное заявкой в первой очереди, которое приблизительно равно половине от максимального.

Для третьей же очереди наибольшее максимальное (и, в частности, среднее) время нахождения в ней заявки связано со сложностью работы соответствующей ей ленты. Она преобразовывает исходный массив, записывая полученные в результате вычитаний и делений новые значения в результирующий массив, на что тратится большое количество операций и, соответственно, время.

Минимальное время нахождения заявки в каждой очереди соответствует отметкам первой задачи: в каждую ленту она попадает сразу же, не ожидая окончания обработки в этом потоке предыдущей задачи.

Вычитая из минимального времени, проведенного заявкой в системе, минимальное суммарное время, проведенное заявкой в очередях, можно вычислить время обработки этой заявки, равное 130 мс.

При этом можно заметить, что минимальное, максимальное и среднее время, проведенное заявкой в системе слабо отличается от тех же замеров для времени, проведенного заявкой в очередях. Это, а также анализ времени, проведенного заявками в третьей очереди, еще раз подтверждает, что при организации параллельного конвейера необходимо разбивать задачу на этапы, схожие по трудоемкости, иначе большую часть времени заявки будут простаивать в очередях.

## 4.4 Вывод из исследовательской части

Таким образом, параллельная организация обработки данных с использованием конвейера работает быстрее, чем линейная обработка. При этом для достижения наилучших показателей необходимо корректно разделять задачу на этапы: так, чтобы время их выполнения было приблизительно равным, иначе большую часть времени заявки будут простаивать в очереди наиболее трудоемкой ленты.

# Заключение

В результате выполнения лабораторной работы была достигнута поставленная цель: был реализован муравьиный алгоритм для решения задачи коммивояжера и приобретены навыки параметризации алгоритмов.

В рамках выполнения работы были выполнены следующие задачи:

- 1) реализован алгоритм полного перебора для решения задачи коммивояжера;
- 2) изучен и реализован муравьиный алгоритм для решения задачи коммивояжера;
- 3) проведена параметризация муравьиного алгоритма на трех классах данных и подобраны оптимальные параметры;
- 4) проведен сравнительный анализ трудоемкостей реализаций.



# Литература

- [1] Ульянов М. В. РЕСУРСНО-ЭФФЕКТИВНЫЕ КОМПЬЮТЕРНЫЕ АЛГОРИТМЫ. РАЗРАБОТКА И АНАЛИЗ // НАУКА ФИЗМАТЛИТ. 2007. С. 201–205.
- [2] Задача коммивояжера [Электронный ресурс]. Режим доступа: <http://www.math.nsc.ru/LBRT/k5/OR-MMF/TSPPr.pdf> (дата обращения: 28.10.2021).
- [3] Алгоритмы решения задачи коммивояжера [Электронный ресурс]. Режим доступа: <https://scienceforum.ru/2021/article/2018025171> (дата обращения: 28.10.2021).
- [4] Стандартное отклонение [Электронный ресурс]. Режим доступа: <https://www.uznaychtotakoe.ru/standartnoe-otklonenie/> (дата обращения: 19.10.2021).
- [5] Thread support library [Электронный ресурс]. Режим доступа: <https://en.cppreference.com/w/cpp/thread> (дата обращения: 20.10.2021).
- [6] Структура systemclock [Электронный ресурс]. Режим доступа: <https://docs.microsoft.com/ru-ru/cpp/standard-library/system-clock-structure?view=msvc-160> (дата обращения: 20.10.2021).