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 Конвейерная обработка данных

Студент Зайцева А.А.

Группа ИУ7-52Б

Преподаватели Волкова Л.Л.

Москва — 2021 г.

Оглавление

Введение

Разработчики архитектуры компьютеров издавна прибегали к методам проектирования, известным под общим названием "совмещение операций при котором аппаратура компьютера в любой момент времени выполняет одновременно более одной базовой операции.

Этот общий метод включает в себя, в частности, такое понятие, как конвейеризация. Конвейры широко применяются программистами для решения трудоемких задач, которые можно разделить на этапы, а также в большинстве современных быстродействующих процессоров [1].

Целью данной работы является изучение организации конвейерной обработки данных на базе алгоритма стандартизации массива.

В рамках выполнения работы необходимо решить следующие задачи:

1) изучить основы конвейеризации;

2) изучить алгоритм стандартизации;

3) разработать параллельную версию конвейера для стандартизации с 3 стадиями обработки;

4) реализовать линейный и параллельный конвейерный варианты стандартизации;

5) собрать статистику времени обработки заявок конвейром;

6) провести сравнительный анализ времени работы реализаций.

1 Аналитическая часть

В данном разделе будет приведена теория, необходимая для разработки и реализации линейного и параллельного конвейерного вариантов стандартизации массива.

1.1 Конвейерная обработка данных

Конвейеризация (или конвейерная обработка) в общем случае основана на разделении подлежащей исполнению функции на более мелкие части, называемые ступенями, и выделении для каждой из них отдельного блока аппаратуры. Так обработку любой машинной команды можно разделить на несколько этапов (несколько ступеней), организовав передачу данных от одного этапа к следующему.

Конвейерную обработку можно использовать для совмещения этапов выполнения разных команд. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько команд [?].

1.2 Стандартизация данных

В широком смысле стандартизация данных представляет собой этап их предобработки с целью приведения к определённому формату и представлению [?].

Стандартизация приводит все исходные значения набора данных, независимо от их начальных распределений и единиц измерения, к набору значений из распределения с нулевым средним и стандартным отклонением, равным 1. В результате формируется так называемая стандартизированная шкала, которая определяет место каждого значения в наборе данных. Значения стандартизированной шкалы определяются по формуле (1.1)

$$z_i = \frac{x_i - X_{mean}}{D_x}, \tag{1.1}$$

где x_i — исходное значение признака, $X_{mean} = \frac{sum(x)}{count(x)}$ и $D_x = \sqrt{\frac{sum((x-X_{mean})^2)}{count(x)}}$ — среднее значение и стандартное отклонение признака, оцененные по набору данных [**?**].

1.3 Вывод из аналитической части

В данном разделе были рассмотрены идеи и материалы, необходимые для разработки и реализации линейного и параллельного конвейерного вариантов стандартизации массива.

2 Конструкторская часть

В данном разделе будет описана структура и принцип работы разрабатываемого конвейера, а также будут приведены схемы для: этапов алгоритма стандартизации данных; линейного алгоритма стандартизации; 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 1 и 2 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 и покидания очередной заявкой каждой очереди.

2.2 Схемы этапов алгоритма стандартизации

На рисунках 2.1 - 2.3 приведены схемы этапов алгоритма стандартизации.

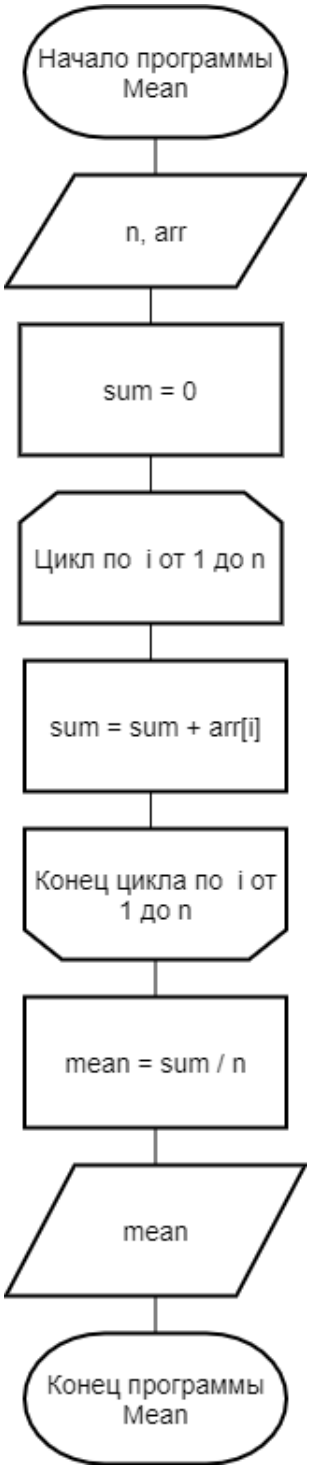

Рисунок 2.1 – Схема этапа поиска среднего значения в массиве

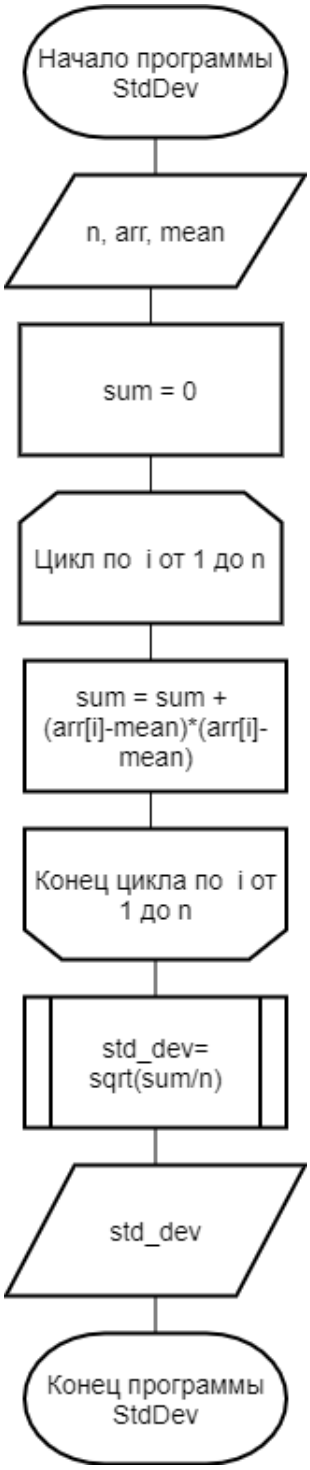

Рисунок 2.2 – Схема этапа поиска стандартного отклонения

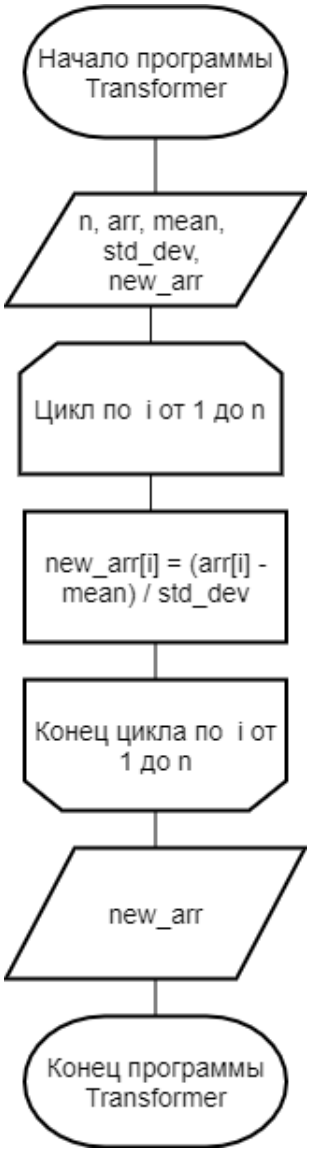

Рисунок 2.3 – Схема этапа преобразования (стандартизации) массива

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 параллельного конвейера для стандартизации

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

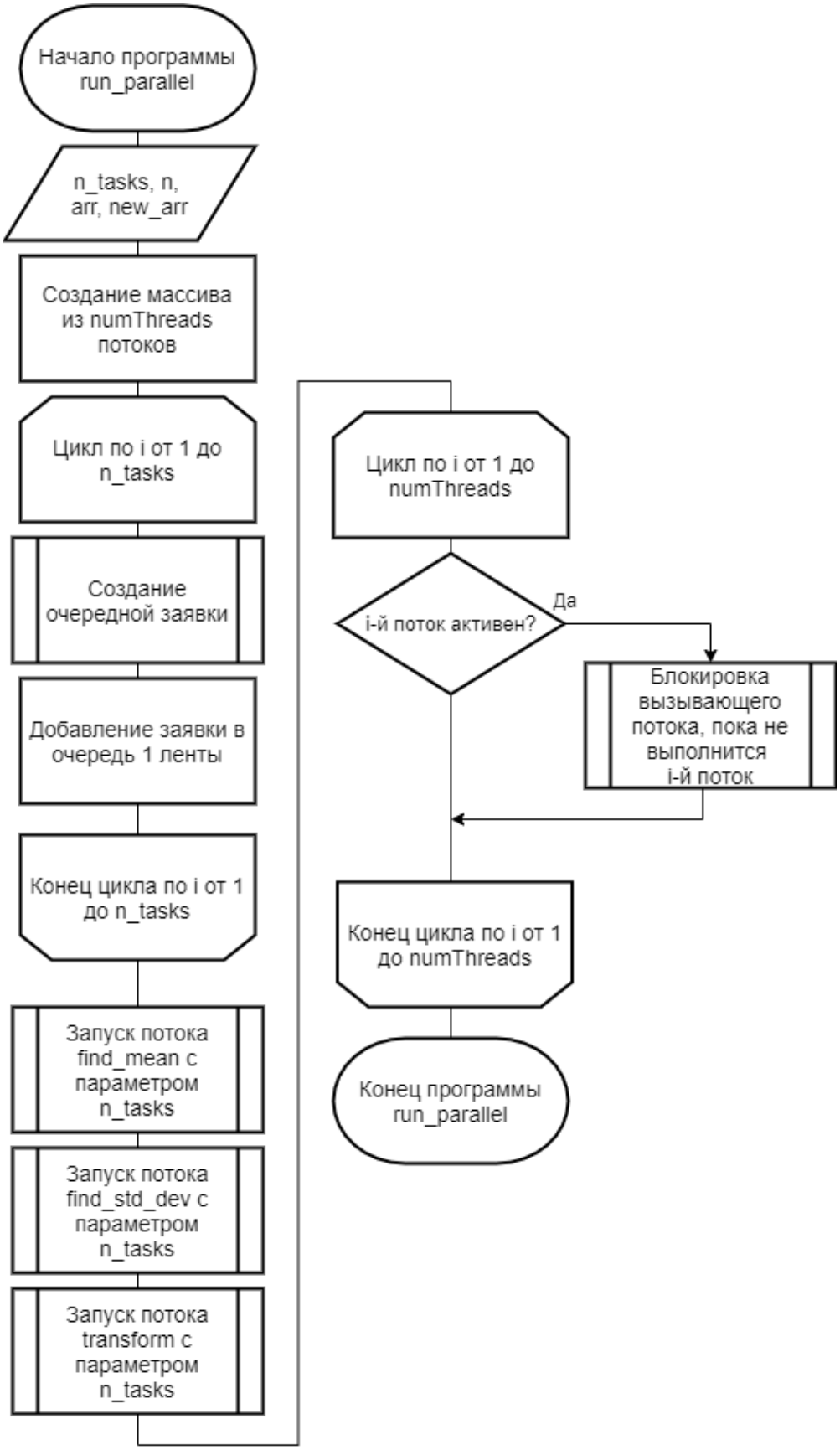

Рисунок 2.5 – Схема главного потока параллельного конвейера

На рисунках 2.6 - 2.8 приведены схемы рабочих потоков конвейера.

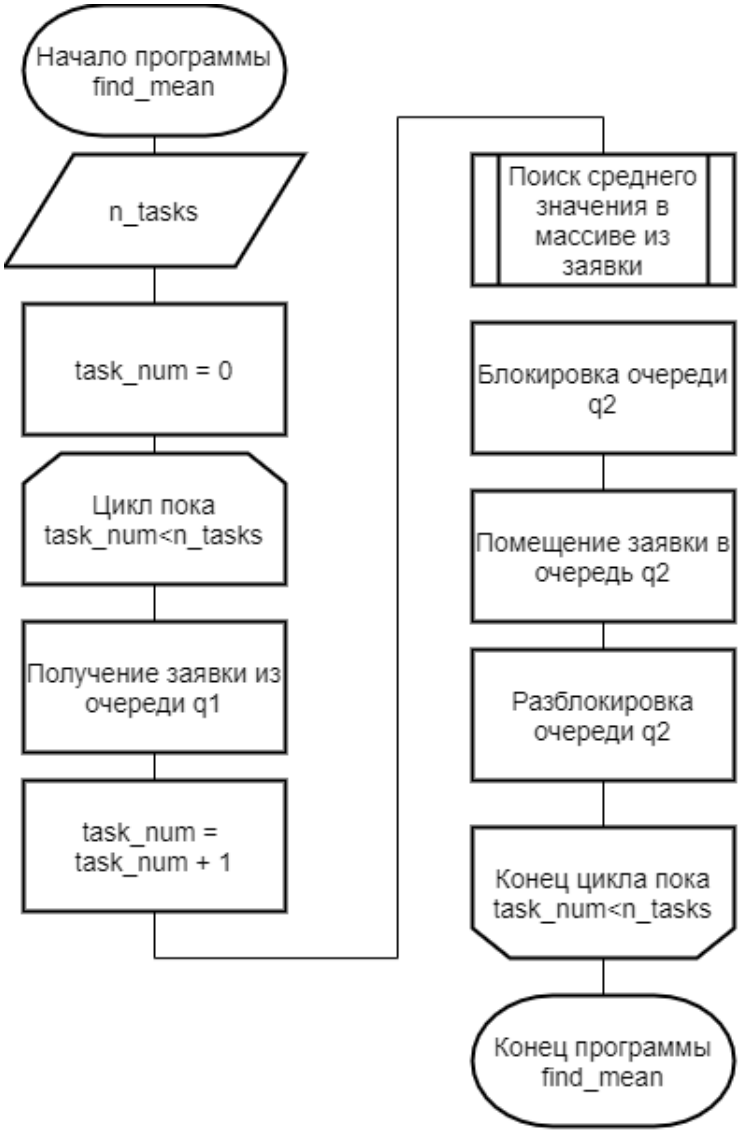

Рисунок 2.6 – Схема потока для поиска среднего значения в массиве

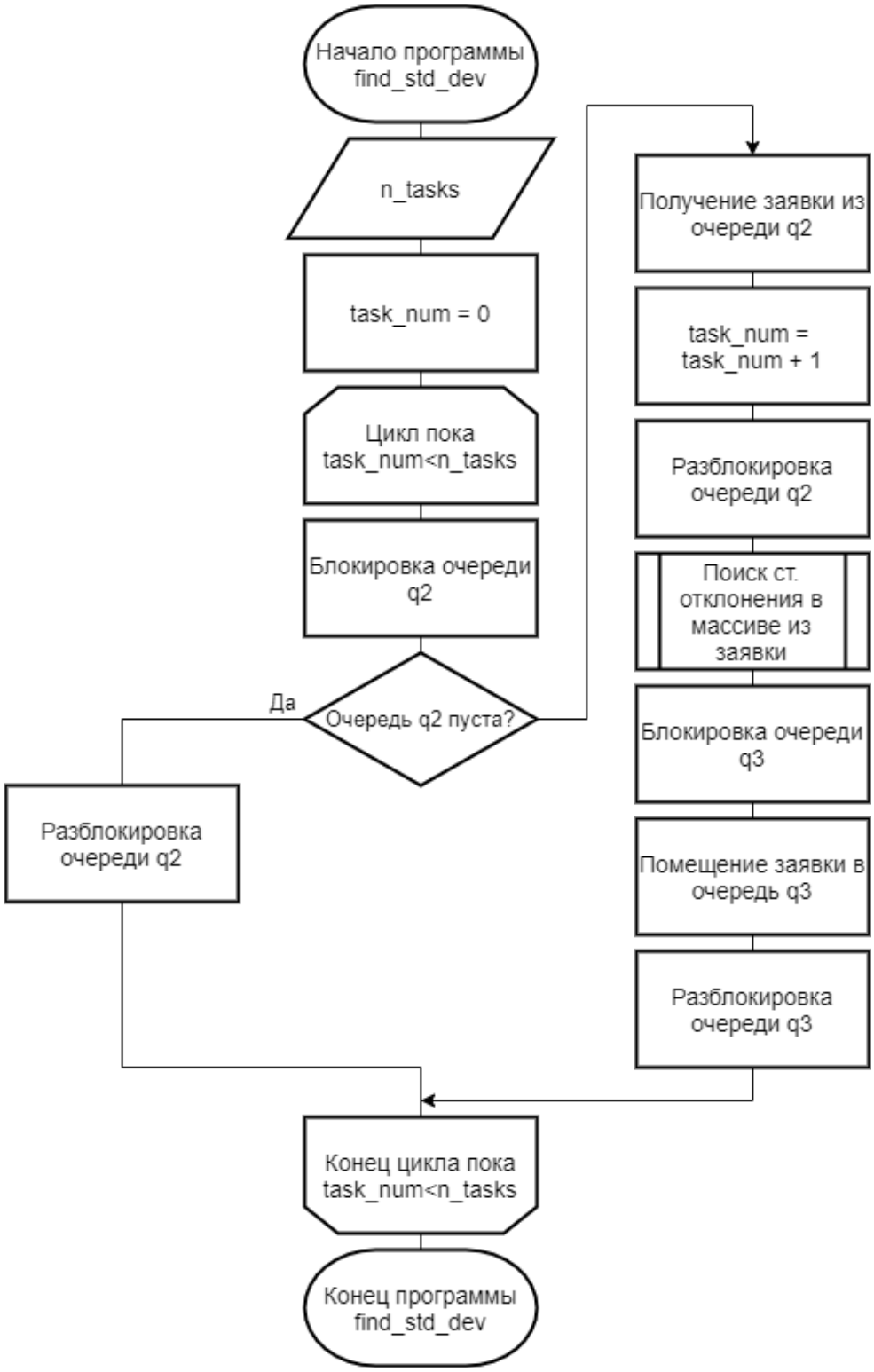

Рисунок 2.7 – Схема потока для поиска стандартного отклонения

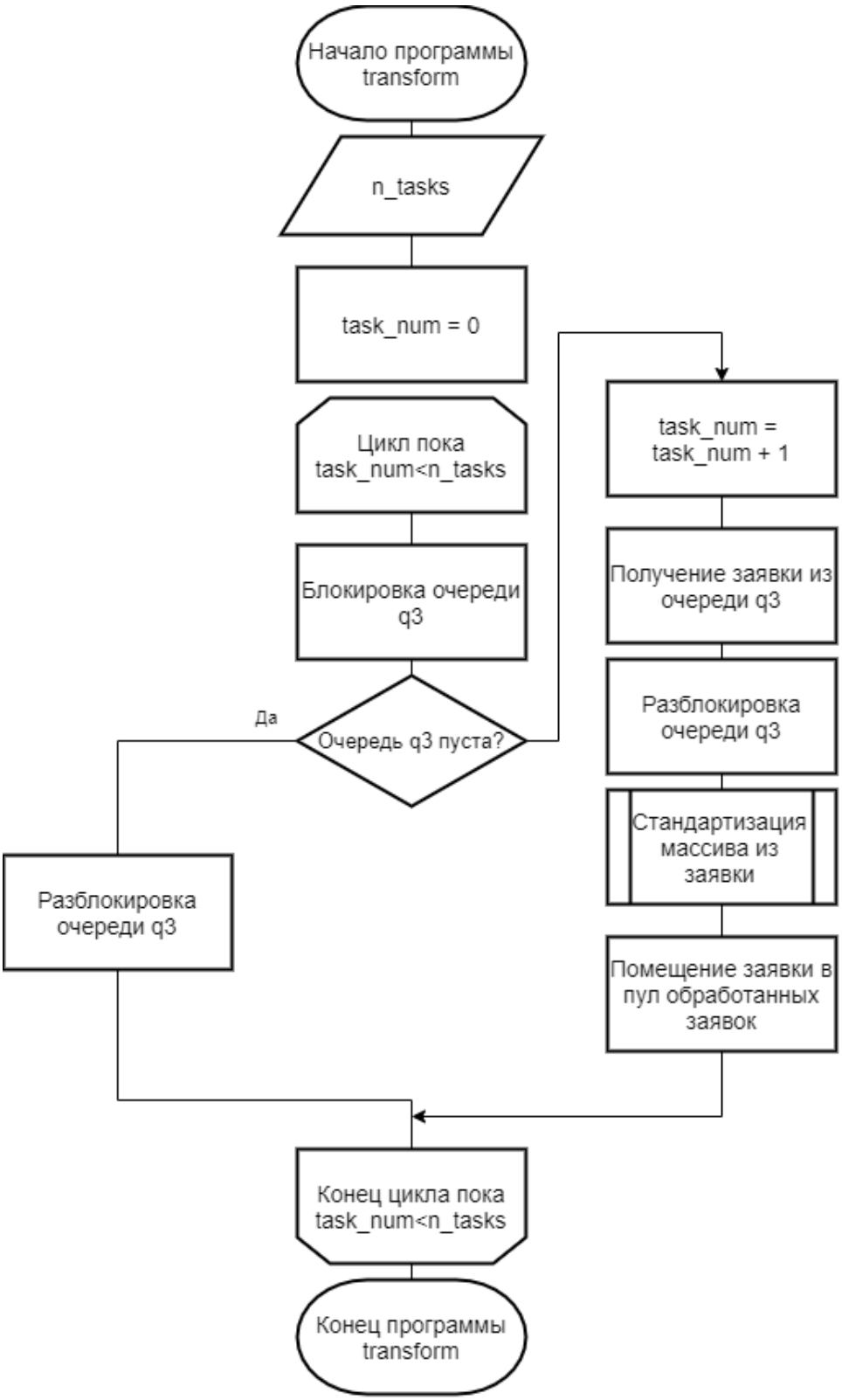

Рисунок 2.8 – Схема потока для преобразования (стандартизации) массива

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 листинги реализованных алгоритмов и 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 и уже использовался мною ранее, поэтому и был выбран [4].

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

В качестве среды разработки выбрана “Visual Studio 2019” так как она предоставляет автоматическую проверку кода и его автоматический рефакторинг, что делает возможным быстрое выявление и исправление ошибок.

3.2 Реализация алгоритмов

В листингах 3.1 - 3.3 представлены реализации рассматриваемых алгоритмов.

Листинг 3.1 – Реализация последовательного алгоритма трассировки лучей

```
1 public Bitmap render_simple()
2 {
3   Camera camera = scene.camera;
4   Vec3 view_vector = null;
5   Vec3 color = null;
6   for (int x = -scene.canvasWidth / 2; x < scene.canvasWidth / 2;
        x++)
7   {
```

```
8       for (int y = -scene.canvasHeight / 2; y < scene.canvasHeight /
            2; y++)
9       {
10        view_vector = ProjectPixel(x, y) * camera.rotation_mtrx;
11        color = TraceRay(camera.position, view_vector,
              projection_plane_d, Double.PositiveInfinity,
              recursion_depth, x, y);
12        SetPixel(x, y, CountColor(color));
13
14      }
15    }
16
17    return this.bmp;
18  }
```

Листинг 3.2 – Реализация главного потока параллельного алгоритма трассировки лучей

```
1   public Bitmap render_parallelized()
2     {
3       Thread[] threads = new Thread[numThreads];
4       for (int i = 0; i < numThreads; i++)
5       {
6         Limits p = new Limits(scene.canvasWidth / numThreads, scene.
              canvasHeight, -scene.canvasWidth / 2 + scene.canvasWidth
              / numThreads * i, -scene.canvasHeight / 2);
7         threads[i] = new Thread(render_par_working);
8         threads[i].Start(p);
9       }
10      foreach (Thread thread in threads)
11      {
12        thread.Join();
13      }
14
15      return this.bmp;
16    }
```

Листинг 3.3 – Реализация рабочего потока параллельного алгоритма трассировки лучей

```
1
2     public void render_par_working(object obj)
3     {
```

```
4       Limits p = (Limits)obj;
5       Camera camera = scene.camera;
6       Vec3 view_vector = null;
7       Vec3 color = null;
8       for (int x = p.start_x; x < p.start_x + p.width; x++)
9       {
10        for (int y = p.start_y; y < p.start_y + p.height; y++)
11        {
12          view_vector = ProjectPixel(x, y) * camera.rotation_mtrx;
13          color = TraceRay(camera.position, view_vector,
                projection_plane_d, Double.PositiveInfinity,
                recursion_depth, x, y);
14          SetPixel(x, y, CountColor(color));
15
16        }
17      }
```

3.3 Тестирование

На рисунке 3.1 приведен результат тестирования программы по методу черного ящика: проведен запуск программного обеспечения для рендера сцены, состоящей из поверхности земли, повехности неба (стена на некотором расстоянии с текстурой) и флюгера (сложносоставной модели), добавлены два источника света – один точечный и один направленный. Программа успешно выполнила рендер сцены на восьми потоках, изображение совпало с ожидаемым – с тем, которое было получено при запуске последовательной версии алгоритма. На рис. 3.1 представлено полученное изображение.

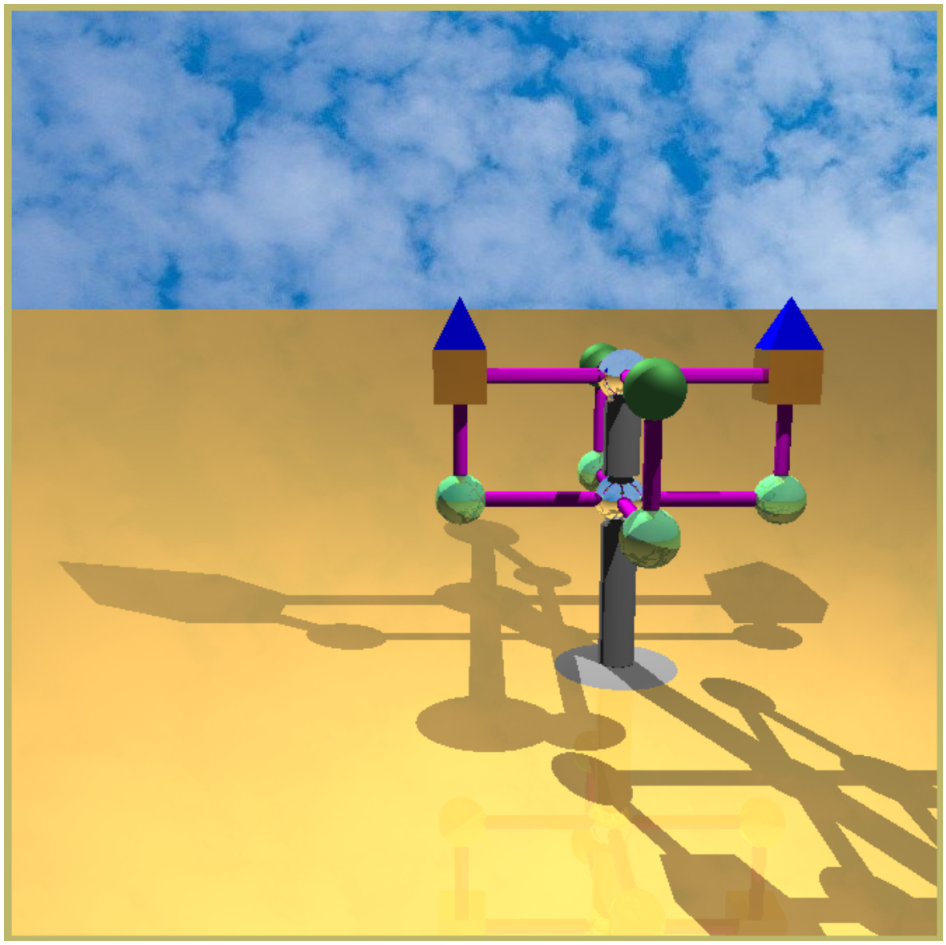

Рисунок 3.1 – Результат тестирования программы

3.4 Вывод из технологической части

Был производен выбор средств реализации, реализованы и протестированы последовательный и параллельный алгоритмы обратной трассировки лучей.

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 и параллельной реализаций алгоритма обратной трассировки лучей, причем во втором случае сравнивалось также время работы реализации в зависимости от количества потоков (1, 2, 4, ..., 4*количесттво логических ядер=32).

Перечисленные реализации сравнивались по времени обработки сцены в зависимости от количества объектов (прямоугольных параллелепипедов) в ней: от 5 до 35 с шагом 5.

Так как некоторые реализации выполняются достаточно быстро, а замеры времени имеют некоторую погрешность, они для каждой реализации

и каждого количества элементов на сцене выполнялись 10 раз, а затем вычислялось среднее время работы.

На рисунке 4.1 приведены результаты сравнения времени работы всех реализаций на всех данных (в легенде количество потоков указано как n_threads).

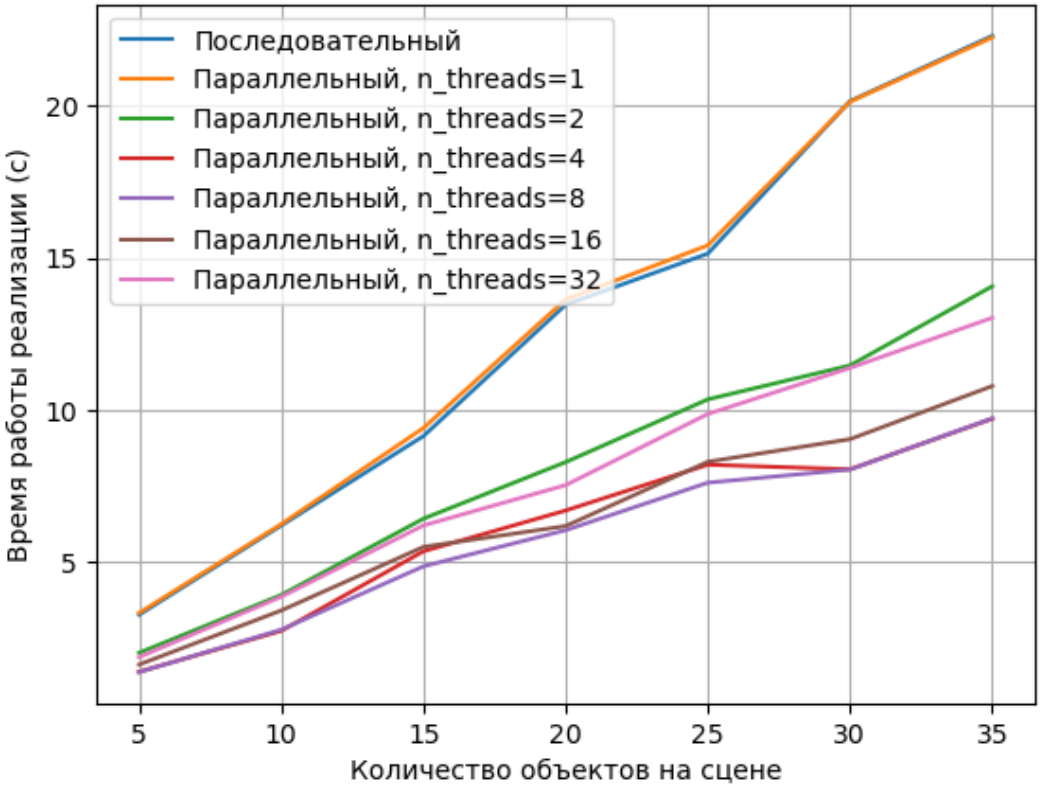

Рисунок 4.1 – Сравнение времени работы реализаций в зависимости от количества элементов на сцене

Последовательная реализация и параллельная реализация с одним потоком, как и ожидалось, работают примерно одинаковое количество времени, хотя при этом вторая немного дольше в связи с накладными расходами на создание потока. Эти две реализации затратили наибольшее количество времени из всех сравниваемых.

Далее с ростом числа потоков время работы соответствующей параллельной реализации уменьшается, так как независимые вычисления производятся одновременно на разных ядрах. Это происходит вплоть до момента, когда используются 8 потоков, то есть их количество равно количеству логических ядер в компьютере.

При дальнейшем увеличении числа потоков время работы параллельной реализации больше, чем в описанной выше наилучшей точке, так как количество потоков становится больше количества логических ядер в компьютере, и, соответственно, некоторые из них вынуждены ожидать освобождения занятого другим потоком процессора, который смог бы провести необходимые вычисления. В результате теряется смысл в выделении этих потоков, так как одновременной обработки каждого из них не происходит, а дополнительное время на их организацию затрачивается.

4.3 Вывод из исследовательской часть

Таким образом, далеко не всегда двукратное увеличение числа потоков улучшает результат по времени. Это происходит, пока количество потоков меньше или равно количеству логических ядер в ЭВМ. Поэтому рекомендуемым числом потоков можно назвать число, равное количеству логических процессоров.

Заключение

В результате выполнения лабораторной работы была достигнута поставленная цель: были изучены основы организации параллельных вычислений на базе алгоритма трассировки лучей.

В ходе выполнения лабораторной работы были выполнены следующие задачи:

1) изучены основы параллельных вычислений;
2) изучен алгоритм трассировки лучей;
3) разработан параллельный алгоритм трассировки лучей;
4) реализованы последовательная и параллельная трассировки лучей;
5) проведен сравнительный анализ времени работы реализаций.

Литература

[1] Принципы конвейерной технологии [Электронный ресурс]. Режим доступа: https://www.sites.google.com/site/shoradimon/18-principy-konvejernoj-tehnologii (дата обращения: 19.10.2021).

[2] Роджерс Д. Математические основы машинной графики. М.: Мир, 1989. Т. 512.

[3] Параллельная обработка в алгоритме визуализации с трассировкой лучей. [Электронный ресурс]. Режим доступа: https://cyberleninka.ru/article/n/parallelnaya-obrabotka-v-algoritmah-vizualizatsii-s-trassirovkoy-lu (дата обращения: 05.10.2021).

[4] Краткий обзор языка C. [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/ (дата обращения: 06.10.2021).