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:

«Конвейерные вычисления»

Студент **Голикова С. М.**

Группа **ИУ7-55Б**

Оценка (баллы)

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

Москва, 2022г.

СОДЕРЖАНИЕ

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 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 основе алгоритма раскраски графа по количеству смежных вершин.

В рамках выполнения работы необходимо решить следующие задачи:

— изучить основы конвейеризации;
— описать алгоритм раскраски графа по количеству смежных вершин;
— разработать параллельную версию конвейера для раскраски графа с 3 стадиями обработки;
— реализовать линейный и параллельный конвейерный варианты раскраски графа;
— провести сравнительный анализ времени работы реализаций.

1 Аналитический раздел

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. При этом 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].

1.2 Раскраска графа по количеству смежных вершин

Раскраска вершин графа $G(V, E)$ в k различных цветов является отображением множества его вершин V во множество $1, 2, \ldots, k, k \in N$, в котором каждый элемент интерпретируется как цвет вершины.

В данной работе рассматривается раскраска графа в зависимости от количества соседей каждой вершины: чем больше смежных вершин у конкретной вершины, тем в более «горячий» цвет она окрашивается.

Вывод

В данном разделе были рассмотрены идеи, необходимые для разработки и реализации линейного и параллельного конвейерного вариантов раскраски графа.

2 Конструкторский раздел

2.1 Разработка конвейера

Алгоритм раскраски графа по количеству смежных вершин можно разделить на 3 этапа:

1) вычисление числа соседей каждого узла;
2) составление описания графа для graphviz;
3) вывод описания графа в файл с расширением dot.

Таким образом, конвейер состоит из 3 лент, каждая из которых выполняет соответствующий этап.

2.2 Разработка алгоритмов

На рисунках 1-3 представлены схемы этапов алгоритма раскраски графа по количеству смежных вершин.

На рисунке 4 приведена схема линейного алгоритма раскраски графа по количеству смежных вершин.

На рисунках 5-8 представлены схемы алгоритма раскраски графа с использованием конвейерных вычислений.

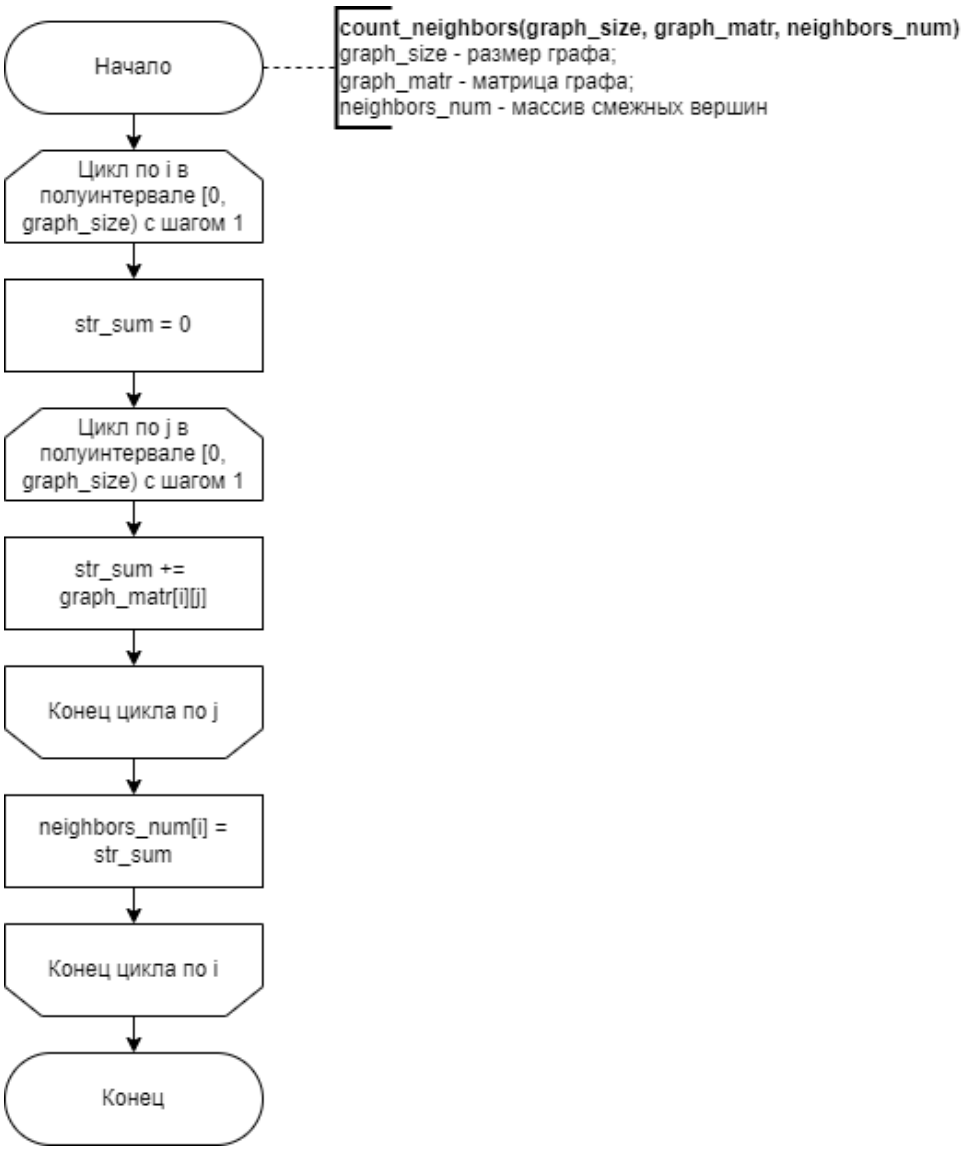

Рисунок 1 – Схема этапа вычисления числа соседей каждого узла

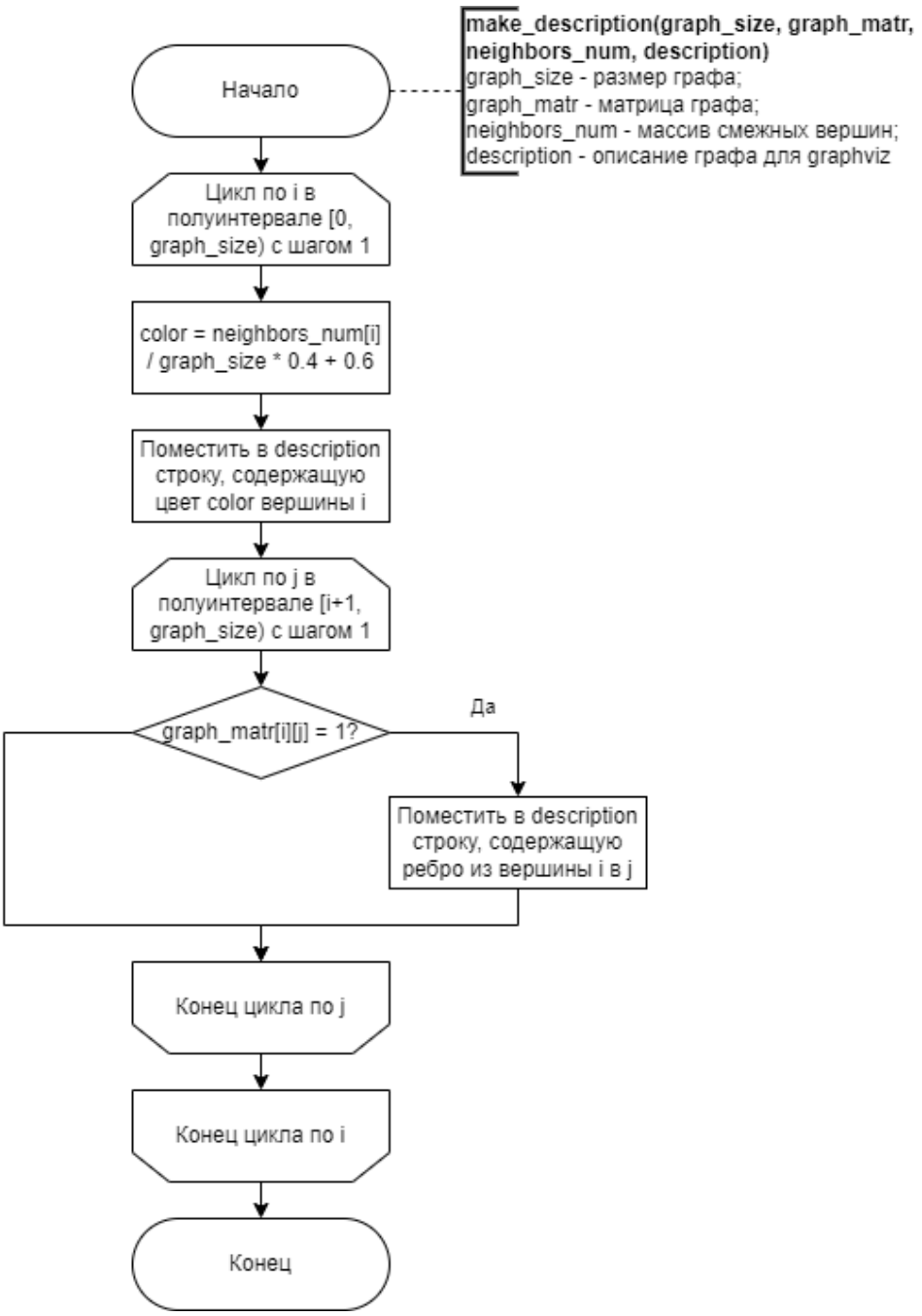

Рисунок 2 – Схема этапа составления описания графа для graphviz

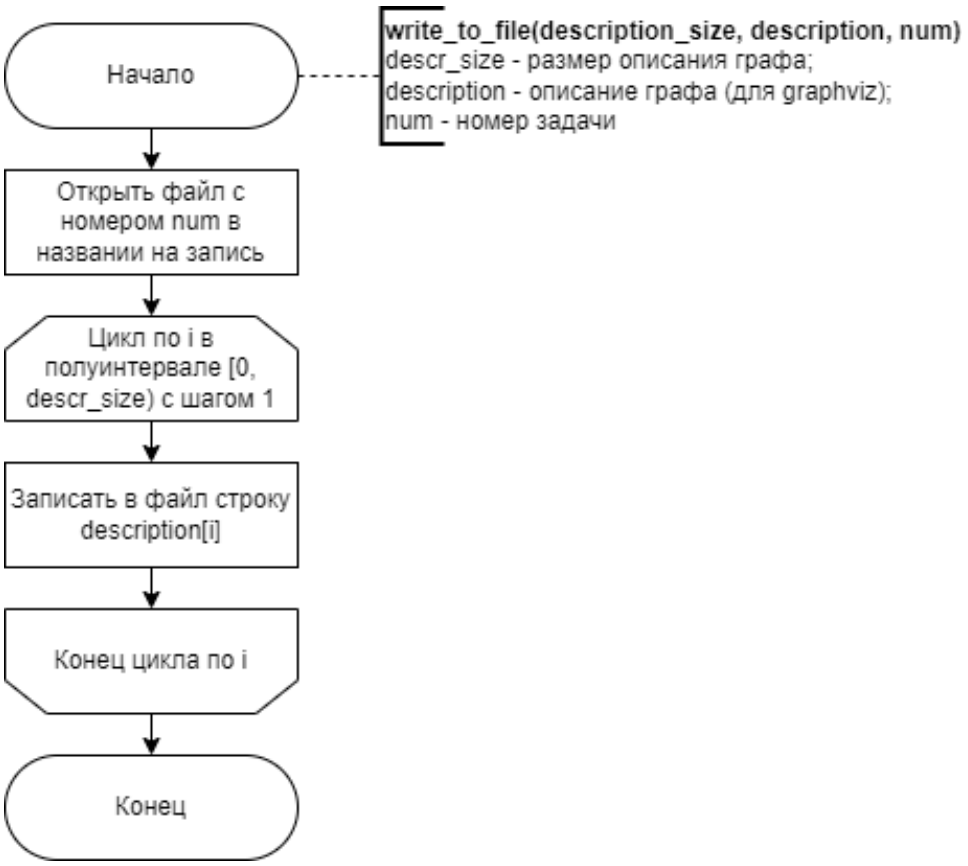

Рисунок 3 – Схема этапа вывода описания графа в файл с расширением dot

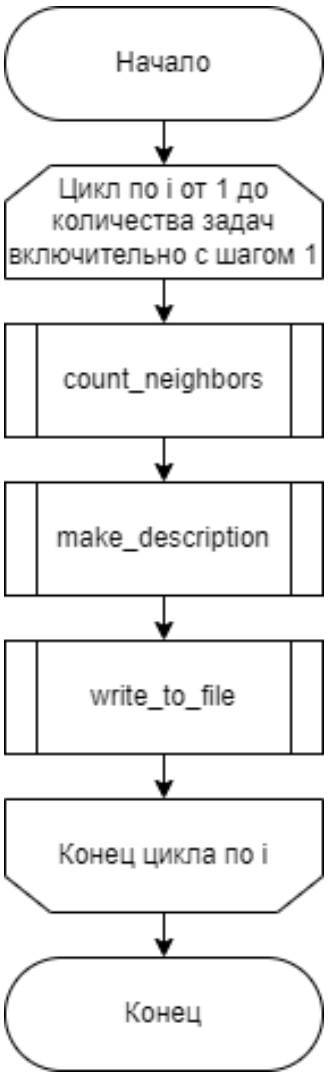

Рисунок 4 – Схема линейного алгоритма раскраски графов

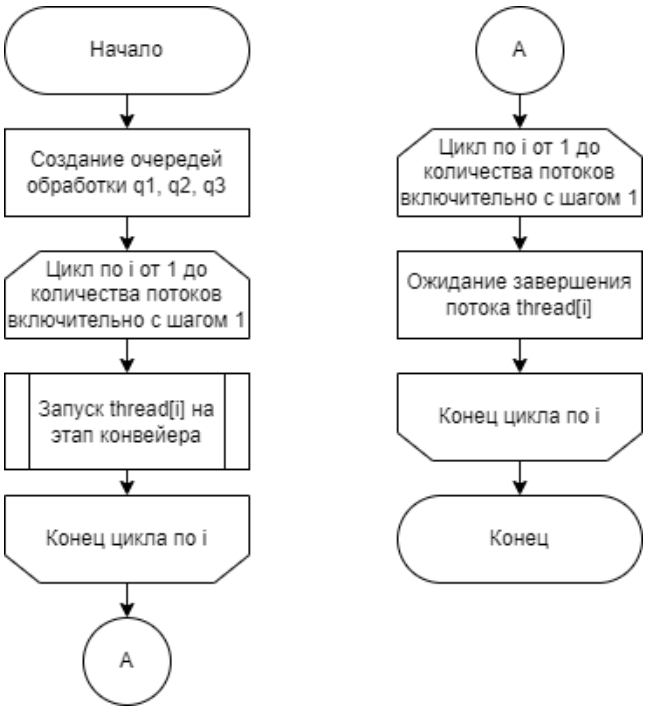

Рисунок 5 – Схема работы главного потока для реализации алгоритма раскраски графа с использованием конвейера

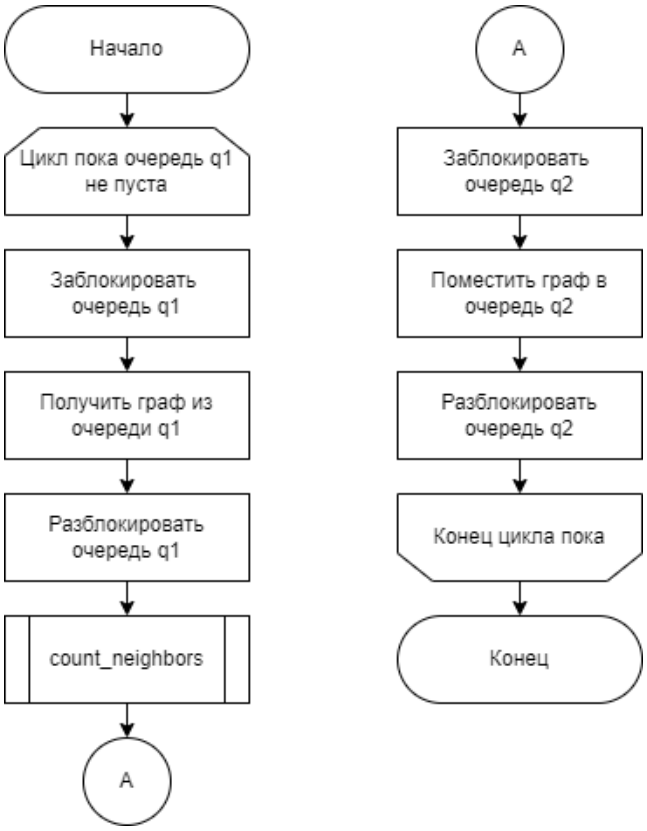

Рисунок 6 – Схема работы потока на первом этапе конвейерных вычислений

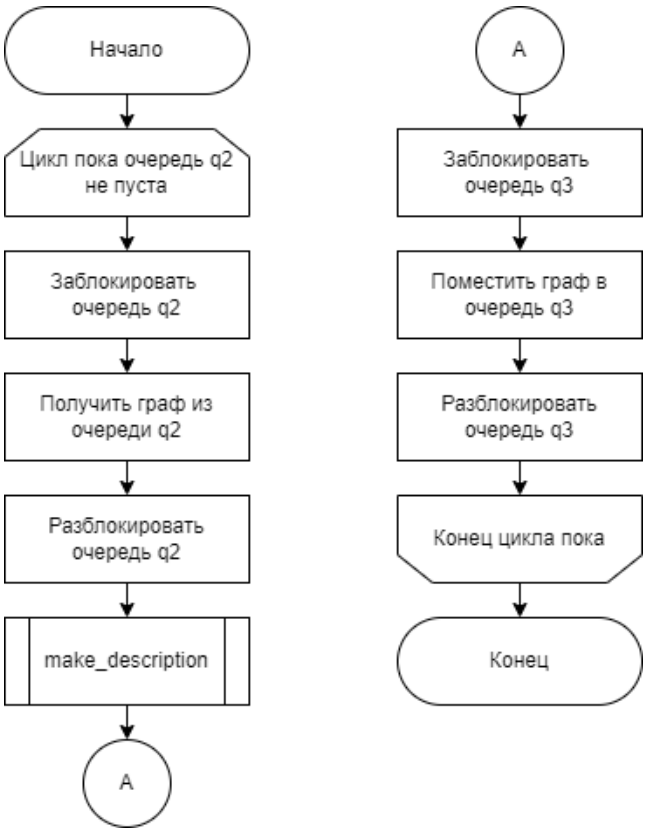

Рисунок 7 – Схема работы потока на втором этапе конвейерных вычислений

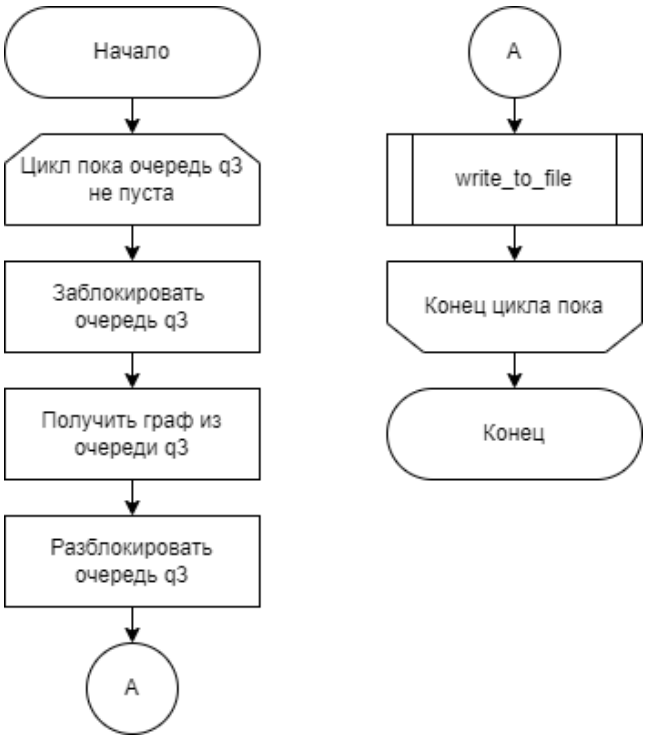

Рисунок 8 – Схема работы потока на третьем этапе конвейерных вычислений

Вывод

Были описаны структура и принцип работы разрабатываемого конвейера, а также приведены схемы разрабатываемых алгоритмов.

3 Технологический раздел

3.1 Требования к программному обеспечению

Программа должна предоставлять следующие возможности:

— выбор режима обработки данных (линейный или конвейерный);

— ввод размера графа и количества графов;

— измерение времени обработки каждой задачи на каждом из этапов;

— вывод результата в файл с расширением dot.

3.2 Средства реализации

В качестве языка программирования для реализации лабораторной работы был выбран язык C++ [2]. Данный язык предоставляет необходимые библиотеки для работы с нативными потоками.

Для визуализации данных эксперимента был выбран язык программирования Python [3], так как он предоставляет большое число настроек параметров графика с использованием простого синтаксиса.

Для замера процессорного времени использовалась функция библиотеки chrono [4] `std::chrono::system_clock::now()`.

3.3 Используемые структуры данных

В программе используется структура graph_t, описанная в листинге 1. Ее полями являются размер графа, матрица со значениями 0 и 1 в зависимости от наличия ребра между вершинами i и j, массив с количеством смежных вершин для каждой вершины, описание графа в формате graphviz.

Листинг 1 – Структура graph_t

```
struct graph_t
{
    int size;
    int **matrix;
    int *neighbors_num;
    std::vector<std::string> description;
};
```

3.4 Реализации алгоритмов

В листингах 2-4 представлены реализации этапов алгоритма раскраски графа по количеству смежных вершин.

В листинге 5 приведена реализация линейного алгоритма раскраски графа по количеству смежных вершин.

В листингах 6 представлена реализация алгоритма раскраски графа с использованием конвейерных вычислений.

Листинг 2 – Этап вычисления числа соседей каждого узла

```
void count_neighbors(graph_t &graph)
{
    for (int i = 0; i < graph.size; i++)
    {
        int str_sum = 0;
        for (int j = 0; j < graph.size; j++)
            str_sum += graph.matrix[i][j];
        graph.neighbors_num[i] = str_sum;
    }
}
```

Листинг 3 – Этап составления описания графа для graphviz

```
void make_description(graph_t &graph)
{
    for (int i = 0; i < graph.size; i++)
    {
        double color = double(graph.neighbors_num[i]) /
                       graph.size * 0.4 + 0.6;

        graph.description.push_back("\t" + std::to_string(i+1)
              + [style=\"filled\", fillcolor=\"" +
              std::to_string(color) + " 1.000 1.000\"]; \n");

        for (int j = i + 1; j < graph.size; j++)
        {
            if (graph.matrix[i][j] == 1)
                graph.description.push_back("\t" + std::to_string
                (i+1) + " -- " + std::to_string(j+1) + ";\n");
        }
    }
}
```

Листинг 4 – Этап вывода описания графа в файл с расширением dot

```
1  void write_to_file(graph_t &graph, int num)
2  {
3      FILE *file = fopen(("output" + std::to_string(num) + ".dot").
           c_str(), "w");
4
5      fprintf(file, "graph { \n");
6
7      for (int i = 0; i < graph.description.size(); i++)
8      {
9          fprintf(file, graph.description[i].c_str());
10     }
11
12     fprintf(file, "}\n");
13
14     fclose(file);
15 }
```

Листинг 5 – Реализация линейного алгоритма

```
1  void stage1_linear(graph_t &graph, int task_num)
2  {
3      count_neighbors(graph);
4  }
5
6  void stage2_linear(graph_t &graph, int task_num)
7  {
8      make_description(graph);
9  }
10
11 void stage3_linear(graph_t &graph, int task_num)
12 {
13     write_to_file(graph, task_num);
14 }
15
16 void parse_linear(int count, size_t size)
17 {
18     std::queue<graph_t> q1, q2, q3;
19     queues_t queues = {.q1 = q1, .q2 = q2, .q3 = q3};
20
21     for (int i = 0; i < count; i++)
22     {
23         graph_t res = generate_graph(size);
24         queues.q1.push(res);
25     }
26
27     for (int i = 0; i < count; i++)
28     {
29         graph_t graph = queues.q1.front();
30         stage1_linear(graph, i + 1);
31         queues.q1.pop();
32         queues.q2.push(graph);
33
```

```
34            graph = queues.q2.front();
35            stage2_linear(graph, i + 1);
36            queues.q2.pop();
37            queues.q3.push(graph);
38
39            graph = queues.q3.front();
40            stage3_linear(graph, i + 1);
41            queues.q3.pop();
42        }
43    }
```

Листинг 6 – Реализация вычислительного конвейера

```
void stage1_parallel(std::queue<graph_t> &q1,
    std::queue<graph_t> &q2, std::queue<graph_t> &q3)
{
    int task_num = 1;

    std::mutex m;

    while(!q1.empty())
    {
        m.lock();
        graph_t graph = q1.front();
        m.unlock();

        count_neighbors(graph);

        m.lock();
        q2.push(graph);
        q1.pop();
        m.unlock();
    }
}

void stage2_parallel(std::queue<graph_t> &q1,
     std::queue<graph_t> &q2, std::queue<graph_t> &q3)
{
    int task_num = 1;

    std::mutex m;

    do
    {
        m.lock();
        bool is_q2empty = q2.empty();
```

```
34          m.unlock();
35
36          if (!is_q2empty)
37          {
38              m.lock();
39              graph_t graph = q2.front();
40              m.unlock();
41
42              make_description(graph);
43
44              m.lock();
45              q3.push(graph);
46              q2.pop();
47              m.unlock();
48          }
49      } while (!q1.empty() || !q2.empty());
50  }
51
52  void stage3_parallel(std::queue<graph_t> &q1,
53      std::queue<graph_t> &q2, std::queue<graph_t> &q3)
54  {
55      int task_num = 1;
56
57      std::mutex m;
58
59      do
60      {
61          m.lock();
62          bool is_q3empty = q3.empty();
63          m.unlock();
64
65          if (!is_q3empty)
66          {
67              m.lock();
68              graph_t graph = q3.front();
```

```
69              m.unlock();
70
71              write_to_file(graph, task_num++);
72
73              m.lock();
74              q3.pop();
75              m.unlock();
76          }
77      } while (!q1.empty() || !q2.empty() || !q3.empty());
78  }
79
80  void parse_parallel(int count, size_t size)
81  {
82      std::queue<graph_t> q1, q2, q3;
83      queues_t queues = {.q1 = q1, .q2 = q2, .q3 = q3};
84
85      for (int i = 0; i < count; i++)
86      {
87          graph_t res = generate_graph(size);
88          q1.push(res);
89      }
90
91      std::thread threads[THREADS];
92
93      threads[0] = std::thread(stage1_parallel,
94              std::ref(q1), std::ref(q2), std::ref(q3));
95      threads[1] = std::thread(stage2_parallel,
96              std::ref(q1), std::ref(q2), std::ref(q3));
97      threads[2] = std::thread(stage3_parallel,
98              std::ref(q1), std::ref(q2), std::ref(q3));
99
100     for (int i = 0; i < THREADS; i++)
101         threads[i].join();
102 }
```

3.5 Тестирование

На рисунке 9 приведен пример исходного графа. На рисунке 10 – ожидаемый результат закраски по количеству смежных вершин для этого графа. Тест пройден успешно.

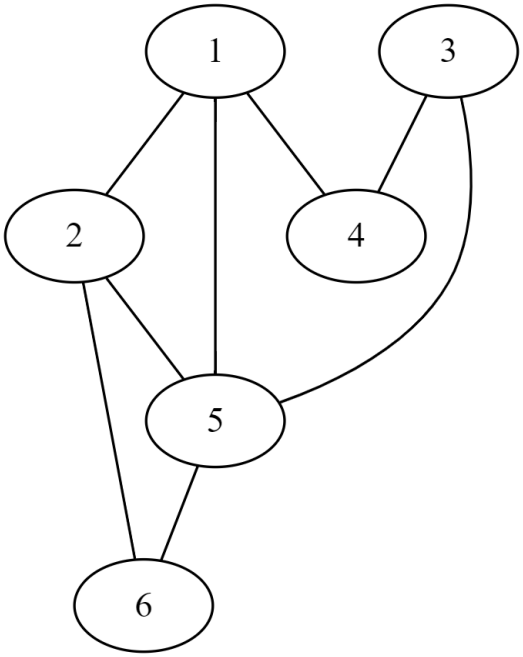

Рисунок 9 – Исходный граф

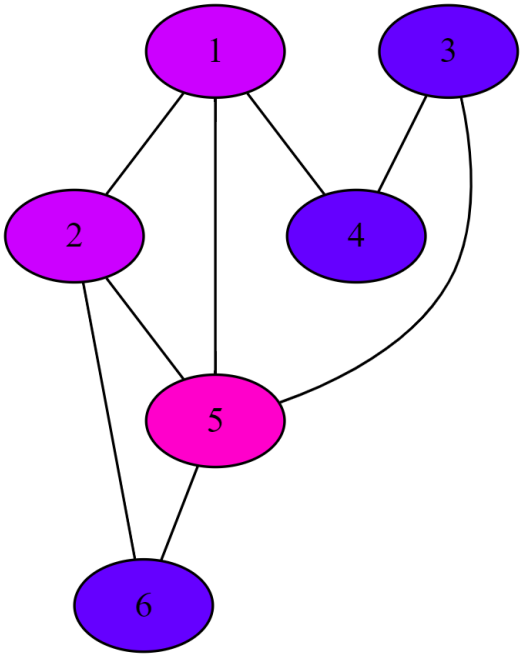

Рисунок 10 – Ожидаемый результат закраски графа

Вывод

Были описаны требования к программному обеспечению, средства реализации, приведены реализации алгоритмов и тест, успешно пройденный программой.

4 Исследовательский раздел

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени:

— операционная система Windows 10;

— память 8 ГБ;

— процессор Intel® Core™ i5-6260U @ 1.80 ГГц, 2 физических ядра, 4 логических ядра.

Замеры времени выполнения реализаций алгоритмов проводились на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно разработанным приложением.

4.2 Демонстрация работы программы

На рисунках 11-12 представлены примеры работы программы для линейной реализации алгоритма и для реализации с использованием вычислительного конвейера соответственно.

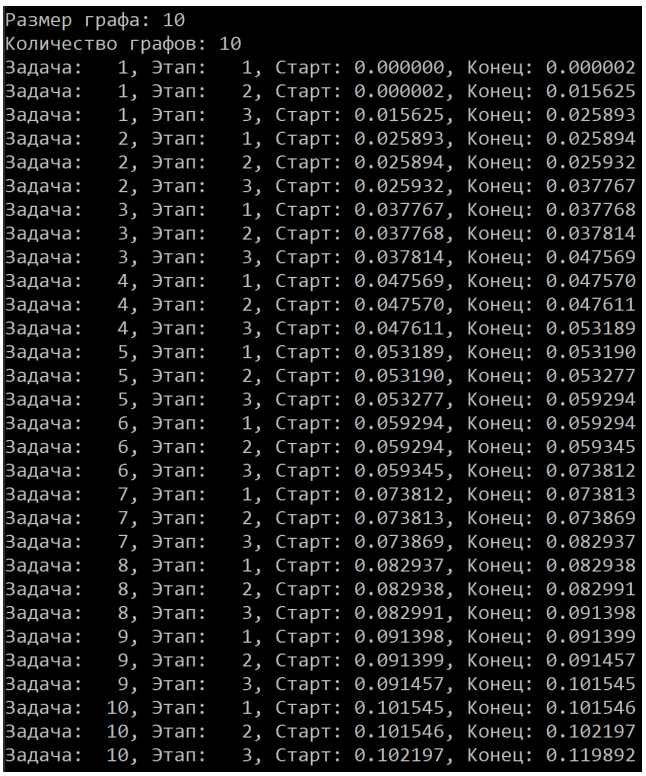

Рисунок 11 – Пример работы линейной реализации алгоритма

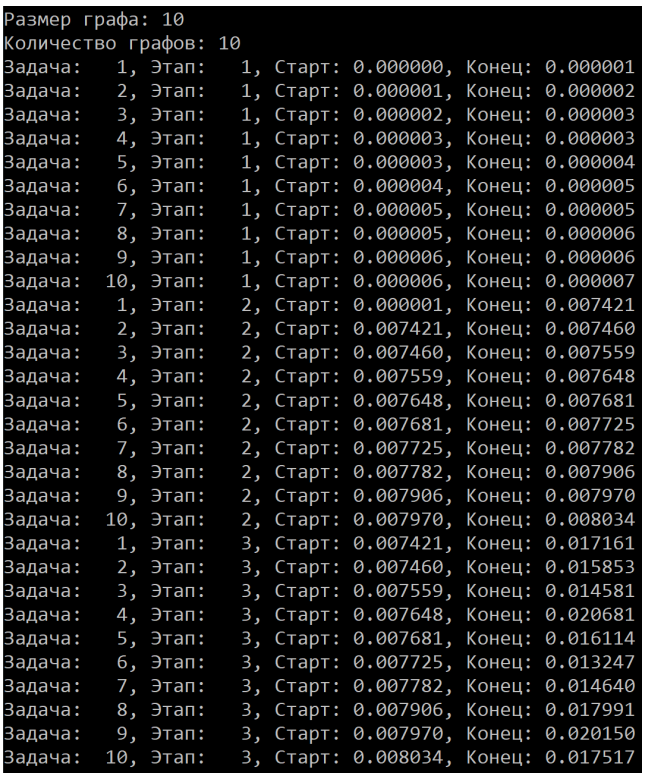

Рисунок 12 – Пример работы реализации с использованием вычислительного конвейера

4.3 Сравнение времени выполнения реализаций алгоритмов

Сравнивается время работы последовательного алгоритма раскраски графа и раскраски графа с использованием параллельного конвейера на графах размером 5 и на количестве задач 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 задач выполняются 10 раз, а затем вычисляется среднее время работы.

В таблице 1 и на рисунке 13 приведены результаты сравнения времени выполнения реализаций алгоритмов.

Таблица 1 – Зависимость времени работы реализации от количества задач

Количество задач	**Линейная реализация (мс)**	**Параллельная реализация (мс)**
1	0.015055	0.010745
5	0.077368	0.011901
25	0.243078	0.036809
50	0.391589	0.093655
100	0.830965	0.314735
150	1.190327	0.414980
200	1.550471	0.619565
250	2.079744	0.856838

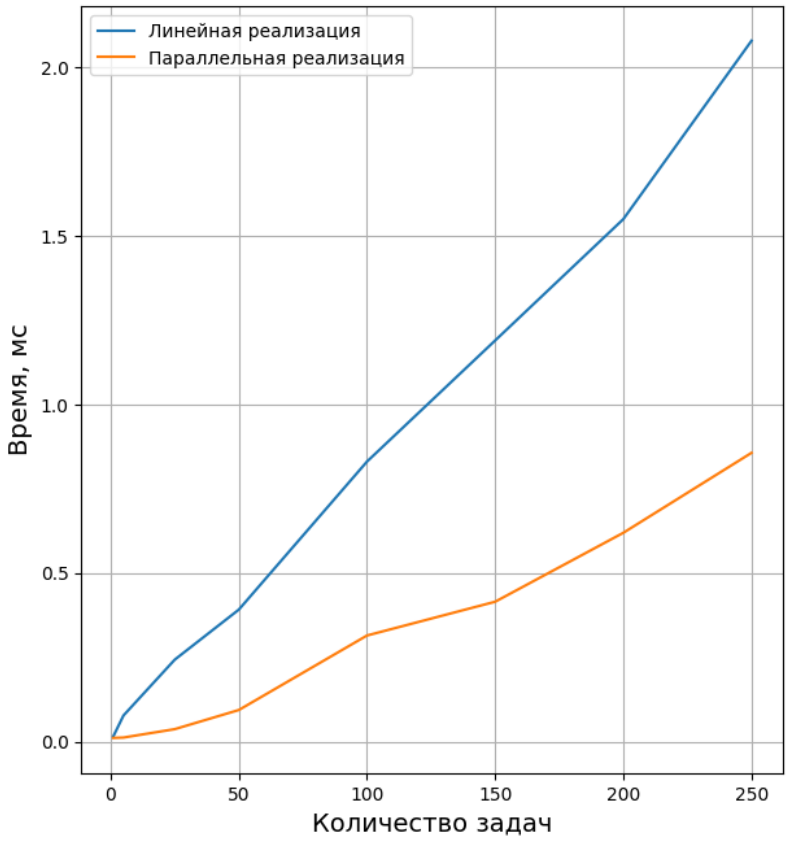

Рисунок 13 – Зависимость времени работы реализации от количества задач

Вывод

Реализация с использованием параллельного конвейера работает быстрее, чем линейная реализация.

Например, при количестве графов 100 параллельная реализация работает в 2.64 раза быстрее, чем линейная, а при количестве 250 – в 2.43 раза. Уже при размере очереди в 5 элементов реализация вычислительного конвейера оправдывает себя.

Таким образом, конвейер позволяет существенно увеличить скорость выполнения задачи, разделив одну задачу на несколько простых.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были решены следующие задачи:

— изучены основы конвейеризации;
— описан алгоритм раскраски графа по количеству смежных вершин;
— разработана параллельная версия конвейера для раскраски графа с 3 стадиями обработки;
— реализованы линейный и параллельный конвейерный варианты раскраски графа;
— проведен сравнительный анализ времени работы реализаций.

Поставленная цель была достигнута: была изучена организация конвейерной обработки данных на основе алгоритма раскраски графа по количеству смежных вершин.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Принципы конвейерной технологии [Электронный ресурс]. Режим доступа: https://www.sites.google.com/site/shoradimon/18-principy-konvejernoj-tehnologii (дата обращения: 01.12.2022).

2. Документация по языку C++ [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/cpp/cpp/?view=msvc-170 (дата обращения: 17.11.2022).

3. Изучаем Python, том 1, 5-е изд / Лутц, Марк. — Пер. с англ. – СПб.: ООО “Диалектика”, 2019. — 832 с.

4. Date and time utilities [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/cpp/chrono (дата обращения: 17.11.2022).