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 № 4 по курсу «Анализ алгоритмов»

Тема Параллельные вычисления на основе нативных потоков

Студент Жаворонкова А. А.

Группа ИУ7-56Б

Оценка (баллы)

Преподаватель Волкова Л. Л.

Москва — 2023 г.

Содержание

Введение

Целью данной работы является исследование параллельных вычислений на основе алгоритма Кнута — Морриса — Пратта. Для достижения поставленной цели необходимо выполнить следующие задачи:

— описать последовательный и параллельный алгоритмы Кнута — Морриса — Пратта;

— реализовать указанные алгоритмы;

— провести тестирование по методу черного ящика для реализаций указанных алгоритмов;

— выполнить сравнительный анализ зависимостей времени решения от размерности входа для реализации параллельного алгоритма;

— описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчетно-пояснительная записка к работе.

1 Аналитическая часть

В данном разделе будут рассмотрены последовательный и параллельный алгоритмы Кнута — Морриса — Пратта.

1.1 Последовательный алгоритм Кнута — Морриса — Пратта

Алгоритм Кнута — Морриса — Пратта используется для поиска специальных подстрок с повторами префиксов [4]. Его основная идея — построение автомата для определения величин смещения. При смещении подстроки после n успешных сравнений и 1 неуспешного считается, что одно сравнение префикса в активе и его не нужно проверять.

Рассмотрим пример. Пусть искомая подстрока — *ababcb*. В построенном автомате состояния маркируются проверяемым в нем символом, дугу — успехом (s) или неудачей (f). Полученный автомат представлен на рисунке 1.1.

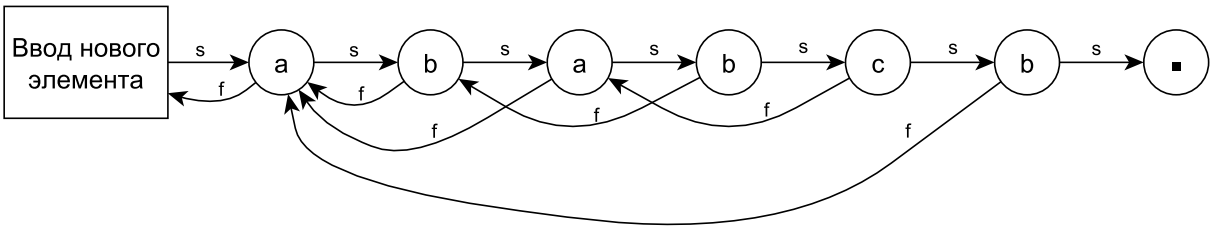

Рисунок 1.1 – Автомат для определения величин смещения

В результате будет получен массив сдвигов, который рассчитывается один раз и используется повторно. Для примера выше таким массивом будет: $[1, 1, 2, 2, 2, 5]$

1.2 Параллельный алгоритм Кнута — Морриса — Пратта

Идея параллельной версии алгоритма Кнута — Морриса — Пратта — разбить строку, в которой выполняется поиск, на n равных частей, где n

— количество потоков. Тогда каждый поток будет искать подстроку в своей части строки.

Однако возможна ситуация, когда исходная строка разделилась таким образом, что части искомой подстроки оказались в разных частях исходной строки, и тогда вхождение не будет найдено. Значит, для корректного решения необходимо делить исходную строку на n частей и к каждой части в начале и в конце добавлять k символов, где k — длина искомой подстроки. В таком случае возможно повторное нахождение подстроки, поэтому при добавлении найденного индекса в результат необходима проверка на уникальность добавляемого значения.

Поскольку n потоков выполняют запись в один результирующий массив найденных индексов, то необходимо средство синхронизации — мьютекс. Мьютексы представляют собой объекты ядра, используемые для синхронизации, регулирующие доступ к единственному ресурсу [5].

1.3 Вывод

В данном разделе были теоретически разобраны последовательный и параллельный алгоритмы Кнута — Морриса — Пратта.

К разрабатываемой программе предъявляются следующие требования.

1. Программа должна предоставлять функциональность поиска подстроки в строке;

2. Реализуемое ПО будет работать в двух режимах — пользовательском, в котором можно выбрать алгоритм и вывести для него результат, а также экспериментальном режиме, в котором можно произвести сравнение реализаций алгоритмов по времени работы;

3. В первом режиме в качестве входных данных в программу будет подаваться файл, содержащий строку, в которой проводится поиск, также реализовано меню для вызова алгоритмов и замеров времени. Программа должна корректно обрабатывать случай ввода подстроки, которой нет в исходной строке;

4. Во втором режиме будет происходить измерение процессорного времени работы программы, будут построены зависимости времени работы от количества рабочих потоков.

2 Конструкторская часть

В этом разделе будут представлены схемы последовательного и параллельного алгоритмов Кнута — Морриса — Пратта.

2.1 Разработка алгоритмов

На рисунках 2.1 — 2.3 представлены схемы схемы последовательного и параллельного алгоритмов Кнута — Морриса — Пратта. В последовательном алгоритме используются мьютексы поскольку впоследствии данная функция используется для параллельного алгоритма.

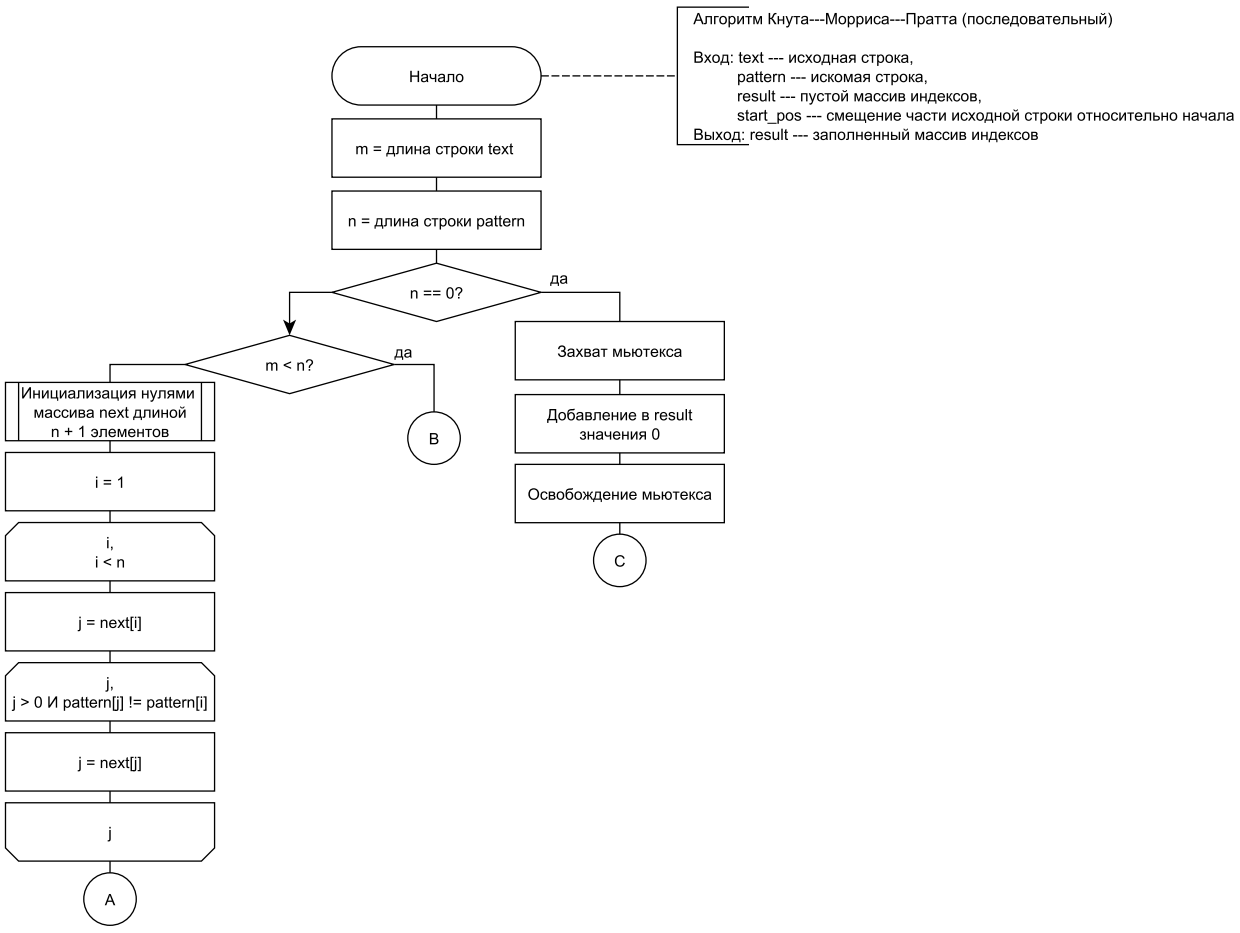

Рисунок 2.1 – Схема последовательного алгоритма Кнута — Морриса — Пратта (часть 1)

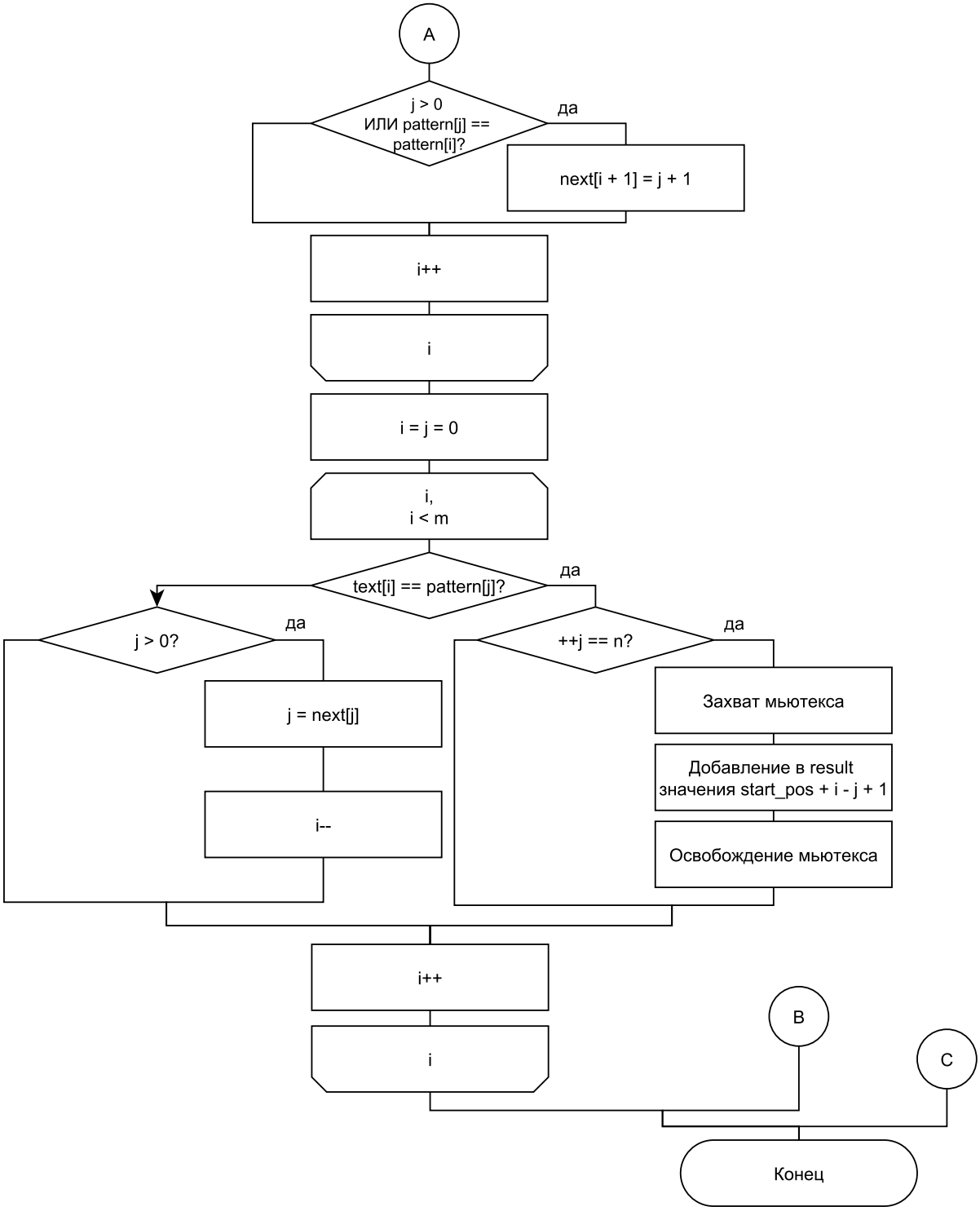

Рисунок 2.2 – Схема последовательного алгоритма Кнута — Морриса — Пратта (часть 2)

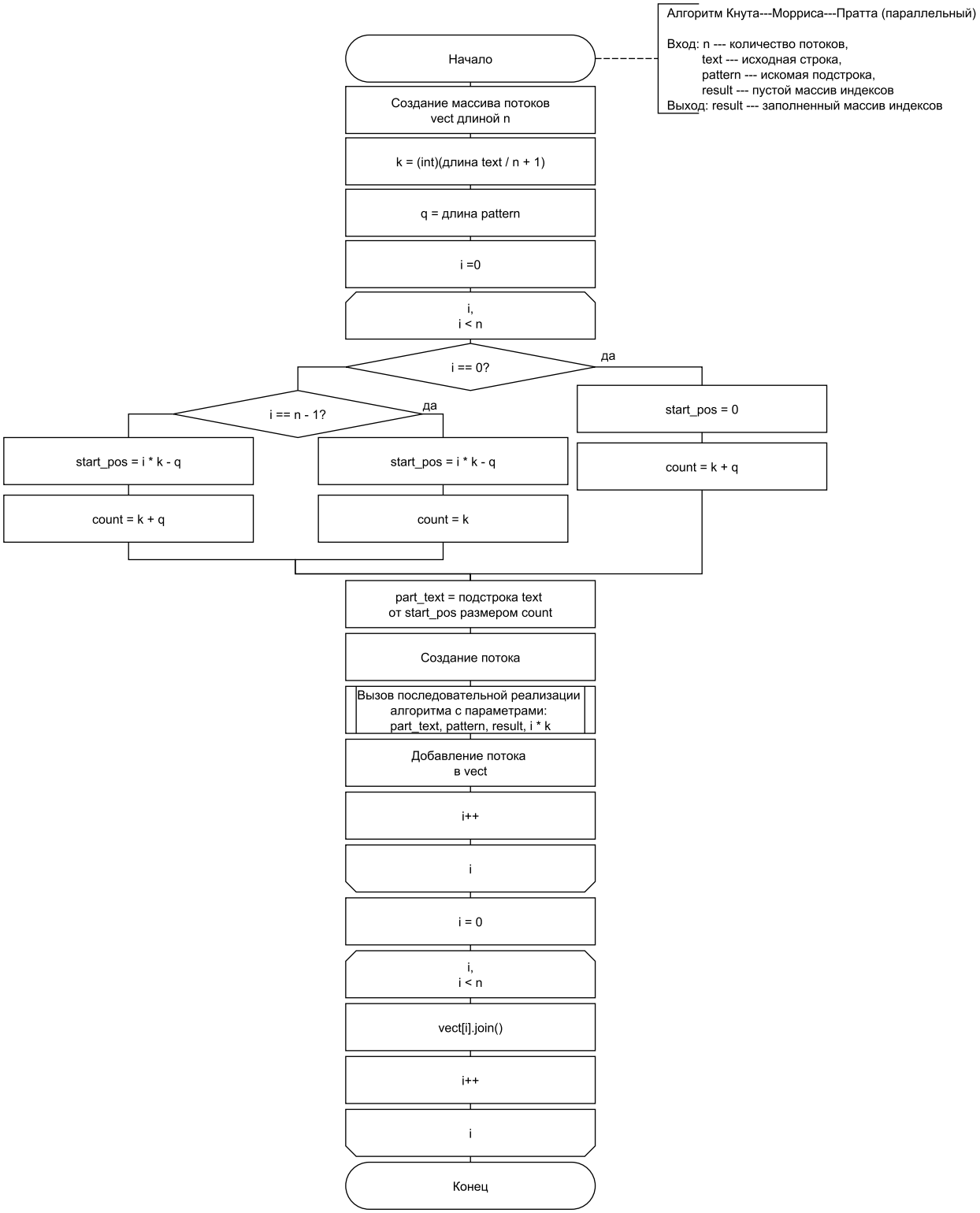

Рисунок 2.3 – Схема параллельного алгоритма Кнута — Морриса — Пратта

2.2 Классы эквивалентности тестирования

Для тестирования выделены следующие классы эквивалентности:

1. ввод подстроки, которой нет в исходной строке;
2. ввод пустой подстроки;
3. ввод подстроки, содержащей повторяющуюся часть;
4. ввод подстроки, состоящей из одного символа;
5. ввод произвольной подстроки.

2.3 Вывод

В данном разделе были представлены схемы алгоритмов, рассматриваемых в лабораторной работе.

3 Технологическая часть

В данном разделе будут рассмотрены средства реализации, а также представлены листинги реализаций последовательного и параллельного алгоритмов Кнута — Морриса — Пратта.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования $C++$ [3]. В текущей лабораторной работе требуется замерить процессорное время для выполняемой программы, а также реализовать принципы многопоточного алгоритма. Все эти инструменты присутствуют в выбранном языке программирования. Время замерено с помощью функции $std :: chrono :: system\_clock :: now()$ из библиотеки *chrono* [2].

3.2 Описание используемых типов данных

При реализации будут использованы следующие типы и структуры данных:

— *std::set* для результирующего набора индексов;

— *std::string* для исходной строки, подстроки;

— *std::thread* для потока;

— *int* для количества потоков.

3.3 Сведения о модулях программы

Программа состоит из одного модуля:

— *main.cpp* — файл, содержащий последовательную и параллельную реализации алгоритмов Кнута — Морриса — Пратта.

3.4 Реализация алгоритмов

В листингах 3.2 — 3.4 представлены последовательная и параллельная реализации алгоритмов Кнута — Морриса — Пратта.

Листинг 3.1 – Реализация последовательного алгоритма Кнута — Морриса — Пратта (начало)

```
1  void KMP(int start_pos, string text, string pattern, std::set<int>
       &result)
2  {
3      int m = text.length();
4      int n = pattern.length();
5      if (n == 0)
6      {
7          mutex.lock();
8          result.insert(0);
9          mutex.unlock();
10         return;
11     }
12     if (m < n)
13         return;
14
15     // next[i] сохраняет индекс следующего лучшего частичного совпад
           ения
16     int next[n + 1];
17     for (int i = 0; i < n + 1; i++) {
18         next[i] = 0;
19     }
20
21     for (int i = 1; i < n; i++)
22     {
23         int j = next[i];
24         while (j > 0 && pattern[j] != pattern[i]) {
25             j = next[j];
26         }
27         if (j > 0 || pattern[j] == pattern[i]) {
28             next[i + 1] = j + 1;
29         }
30     }
```

Листинг 3.2 – Реализация последовательного алгоритма Кнута — Морриса — Пратта (окончание)

```
    for (int i = 0, j = 0; i < m; i++)
    {
        if (text[i] == pattern[j])
        {
            if (++j == n) {
                mutex.lock();
                result.insert(start_pos + i - j + 1);
                mutex.unlock();
            }
        }
        else if (j > 0)
        {
            j = next[j];
            i--;    // так как `i` будет увеличен на следующей итера
               ции
        }
    }
}
```

Листинг 3.3 – Реализация параллельного алгоритма Кнута — Морриса — Пратта (начало)

```
void KMP_par(int n, string text, string pattern, std::set<int>
   &result)
{
    std::vector<std::thread> vect;
    int k = (int)(text.length() / n);
    int q = pattern.length();
    for (int i = 0; i < n; ++i)
    {
        int start_pos, count;
        if (i == 0) {
            start_pos = 0; count = k + q;
        }
        else if (i == n - 1) {
            start_pos = i * k - q; count = k;
        }
        else {
            start_pos = i * k - q; count = k + q;
```

Листинг 3.4 – Реализация параллельного алгоритма Кнута — Морриса — Пратта (окончание)

```
            start_pos = i * k - q; count = k + q;
        }
        if (start_pos < 0)
            start_pos = 0;
        if (start_pos + count > text.length()) {
            start_pos = i * k - q; count = k;
        }

        string part_text = text.substr(start_pos, count);
        std::thread t(KMP, start_pos, part_text, pattern,
           std::ref(result));
        vect.push_back(std::move(t));
    }

    for (int i = 0; i < n; ++i)
        vect[i].join();
}
```

3.5 Функциональные тесты

В таблице 3.1 приведены тесты для функций, реализующих последовательный и параллельный алгоритмы Кнута — Морриса — Пратта. Тесты *для всех алгоритмов* пройдены успешно.

Таблица 3.1 – Функциональные тесты

Исходная строка	Искомая подстрока	Последовательная реализация	Параллельная реализация
abcd	"пустая строка"	0	0
abab	cd	«подстрока не найдена»	«подстрока не найдена»
abababcb	ababcb	2	2
abcabd	abc	0	0
ababcacab	ab	0, 2, 7	0, 2, 7

3.6 Вывод

Были представлены последовательная и параллельная реализации алгоритмов Кнута — Морриса — Пратта, которые были описаны в предыдущем разделе. Также в данном разделе была приведена информация о выбранных средствах для разработки алгоритмов.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программы, а также проведен сравнительный анализ алгоритмов при различных ситуациях на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование представлены далее:

— операционная система: Windows 11, x64;

— оперативная память: 8 Гб;

— процессор: AMD Ryzen 5 5500U с видеокартой Radeon Graphics 2.10 ГГц [1];

— 6 физических ядер, 12 логических ядер [1].

Во время замеров времени ноутбук был нагружен только встроенными приложениями окружения.

4.2 Демонстрация работы программы

На рисунках 4.1–4.2 представлены результаты работы программы.

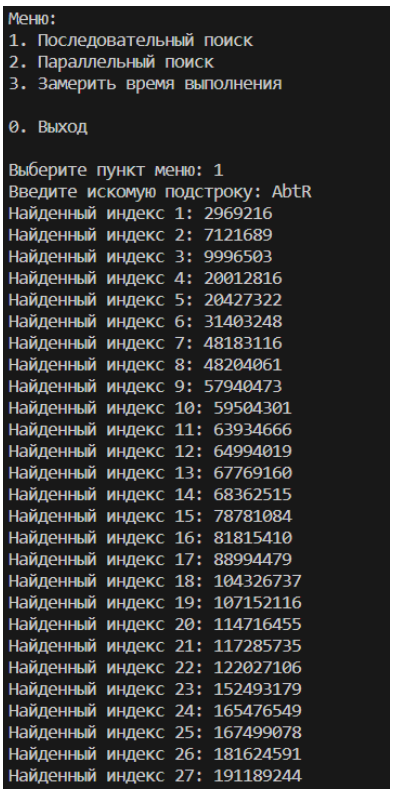

Рисунок 4.1 – Демонстрация работы программы (последовательный поиск)

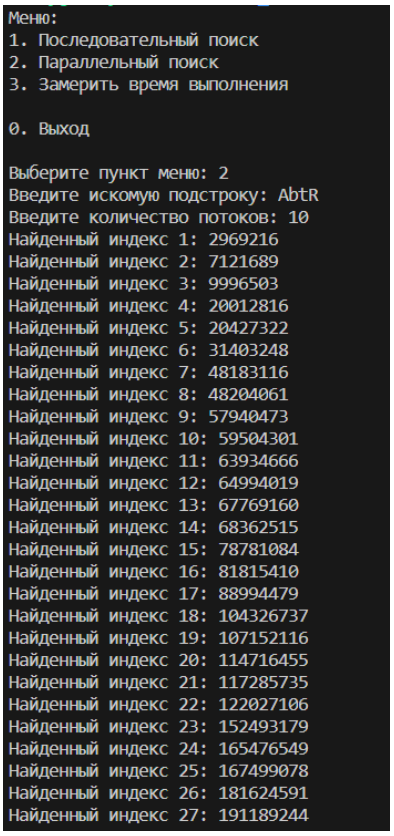

Рисунок 4.2 – Демонстрация работы программы (параллельный поиск)

4.3 Время выполнения алгоритмов

Как было сказано выше, используется функция замера процессорного времени $std :: chrono :: system\_clock :: now()$. Функция возвращает пользовательское процессорное время типа *float*.

Использовать функцию приходится дважды, затем из конечного времени нужно вычесть начальное, чтобы получить результат.

Замеры проводились для количества потоков:

$$\{1, 2, 4, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96\}$$

по 100 раз.

Результаты замеров приведены в таблице 4.1 (время в мс).

Таблица 4.1 – Результаты замеров времени

Количество потоков	Время, мс
1	1201.250
2	683.155
4	443.653
8	279.936
16	235.230
24	228.121
32	216.056
40	224.693
48	233.493
56	225.752
64	242.985
72	237.691
80	225.717
88	244.182
96	221.405

По результатам замеров 32 потока является оптимальным количеством.

В таблице 4.2 приведены результаты замеров времени для последовательной и параллельной реализаций алгоритма Кнута — Морриса — Пратта для различных длин строк. В параллельной реализации использовались 32 потока.

Таблица 4.2 – Результаты замеров времени

Длина строки	Последовательная реализация, мс	Параллельная реализация, мс
100 000	0.512	3.522
200 000	0.946	3.509
300 000	1.377	3.488
400 000	1.864	3.582
500 000	2.322	3.556
600 000	2.729	3.467
700 000	3.219	3.435
800 000	3.621	3.521
900 000	4.065	3.572
1 000 000	4.533	3.435
1 100 000	4.964	3.517
1 200 000	5.434	3.635
1 300 000	5.883	3.809
1 400 000	6.378	3.915
1 500 000	6.718	3.681
1 600 000	7.198	3.913
1 700 000	7.594	3.858
1 800 000	8.138	4.041
1 900 000	8.625	4.120
2 000 000	8.943	4.005

На рисунке 4.3 приведена визуализация результатов замеров.

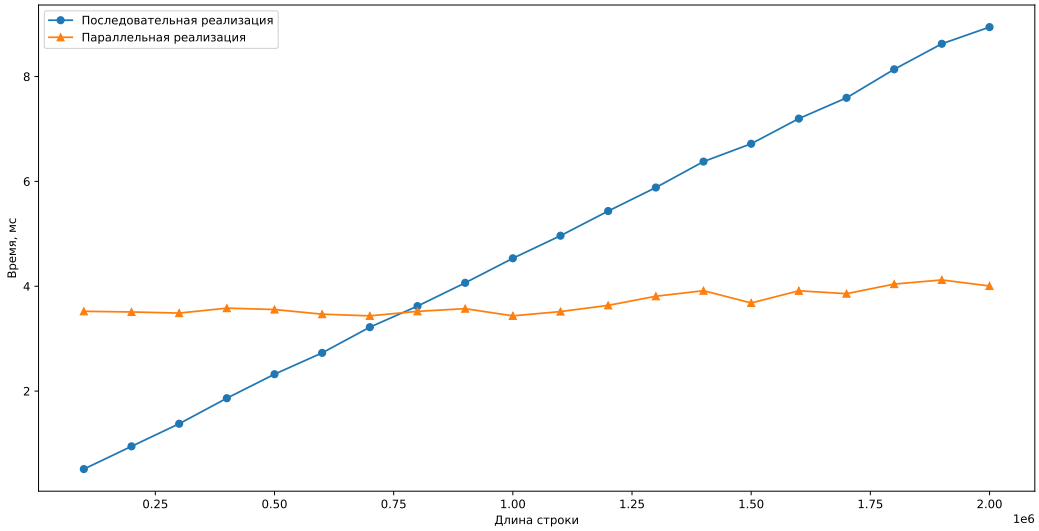

Рисунок 4.3 – Визуализация результатов замеров

4.4 Вывод

Как видно из графика 4.3, время выполнения последовательной реализации линейно зависит от длины исходной строки, а время выполнения параллельной реализации не зависит. Также видно, что при длине исходной строки менее 800 000 последовательная реализация будет работать быстрее параллельной.

Заключение

В результате было получено, что 32 потока является оптимальным количеством для использованной длины исходной строки. Также было установлено, что в параллельной реализации с фиксированным количеством потоков нет зависимости от длины исходной строки, а в последовательной есть — линейная. При длине исходной строки выше 800 000 параллельная реализация с 32 потоками будет выполняться быстрее последовательной.

Цель, которая была поставлена в начале лабораторной работы была достигнута: исследованы параллельные вычисления на основе алгоритма Кнута — Морриса — Пратта. В ходе выполнения лабораторной работы были решены все задачи:

— описаны последовательный и параллельный алгоритмы Кнута — Морриса — Пратта;

— реализованы указанные алгоритмы;

— проведено тестирование по методу черного ящика для реализаций указанных алгоритмов;

— выполнен сравнительный анализ зависимостей времени решения от размерности входа для реализации параллельного алгоритма;

— описаны и обоснованы полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчетно-пояснительная записка к работе.

Список используемых источников

1. AMD Ryzen 5 5500U. — URL: https://www.amd.com/en/products/apu/amd-ryzen-5-5500u (дата обр. 01.12.2023).
2. C++ chrono library. — URL: https://cplusplus.com/reference/chrono/?kw=chrono (дата обр. 01.12.2023).
3. C++ documnetation. — URL: https://cplusplus.com/ (дата обр. 01.12.2023).
4. Математические основы алгоритмов. Строковые алгоритмы. Поиск в строке: алгоритм Кнута–Морриса–Пратта, его реализация на конечном автомате. — URL: https://users.math-cs.spbu.ru/~okhotin/teaching/algorithms1_2022/okhotin_algorithms1_2022_l6.pdf (дата обр. 01.12.2023).
5. НОУ ИНТУИТ. Лекция. Синхронизация потоков. — URL: https://new2.intuit.ru/studies/professional_skill_improvements/1717/courses/217/lecture/5599?page=2&ysclid=lpwnduox8g513624798 (дата обр. 01.12.2023).