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"

Студент Недолужко Денис Вадимович

Группа ИУ7-53Б

Название предприятия МГТУ им. Н. Э. Баумана, каф. ИУ7

Тема Параллельное лексикографическое сравнение строк

Студент _____ Недолужко Д. В.
(Подпись, дата) (И.О. Фамилия)

Руководитель курсовой работы _____ Волкова Л. Л.
(Подпись, дата) (И.О. Фамилия)

Москва
2021 г.

Содержание

Введение

В данной лабораторной работе будет рассмотрен последовательный и параллельный алгоритм лексикографического сравнения строк.

Многопоточность – способность центрального процессора (ЦПУ) или одного ядра в многоядерном процессоре одновременно выполнять несколько процессов или потоков, соответствующим образом поддерживаемых операционной системой.

Этот подход отличается от многопроцессорности, так как многопоточность процессов и потоков совместно использует ресурсы одного или нескольких ядер: вычислительных блоков, кэш-памяти ЦПУ или буфера перевода с преобразованием.

В тех случаях, когда многопроцессорные системы включают в себя несколько полных блоков обработки, многопоточность направлена на максимизацию использования ресурсов одного ядра, используя параллелизм на уровне потоков, а также на уровне инструкций.

Поскольку эти два метода являются взаимодополняющими, их иногда объединяют в системах с несколькими многопоточными ЦП и в ЦП с несколькими многопоточными ядрами.

Многопоточная парадигма стала более популярной с конца 1990-х годов, поскольку усилия по дальнейшему использованию параллелизма на уровне инструкций застопорились.

Целью работы является исследование паралельного алгоритма лексикографического сравнения строк.

Для достижения поставленной цели необходимо выполнить следующие задачи:

а) Изучение алгоритмов лексикографического сравнения строк

б) Реализация последовательного алгоритма лексикографического сравнения строк

в) Реализация параллельного алгоритма лексикографического сравнения строк

г) Экспериментальное сравнение временных характеристик реализованных алгоритмов

1 Аналитическая часть

В данном разделе описаны математические модели исследуемой области.

1.1 Описание задачи

Лексикографический порядок — отношение линейного порядка на множестве слов над некоторым упорядоченным алфавитом $\sum$. Своё название лексикографический порядок получил по аналогии с сортировкой по алфавиту в словаре.

Определение

Слово α предшествует слову β ($\alpha < \beta$), если

— либо первые m символов этих слов совпадают, а $m + 1$-й символ слова α меньше (относительно заданного в $\sum$ порядка) $m+1$-го символа слова β (например, АБАК $<$ АБРАКАДАБРА, так как первые две буквы у этих слов совпадают, а третья буква у первого слова меньше, чем у второго);

— либо слово α является началом слова β (например, МАТЕМАТИК $<$ МАТЕМАТИКА).

Примеры

Порядок слов в словаре. Предполагается, что буквы можно сравнивать, сравнивая их номера в алфавите. Например, следующие слова идут в лексикографическом порядке: А $<$ АА $<$ ААА $<$ ААБ $<$ ААВ $<$ АБ $<$ Б $< \ldots <$ ЯЯЯ.

Естественный порядок на неотрицательных целых n-значных числах в любой позиционной системе счисления, записанных в разрядной сетке фиксированной длины (000, 001, 002, 003, 004, 005, . . . , 998, 999).

Вывод

В результате были рассмотрены и описаны математические модели следующих алгоритмов:

— Последовательный алгоритм лексикографического сравнения строк

— Параллельный алгоритм лексикографического сравнения строк

В качестве входных данных алгоритмы принимают 2 строки. Выходными данными алгоритма является число. Если число равно нулю, то строки равны, если число больше нуля, то первая строка больше второй иначе вторая строка больше первой.

Ограничением для реализуемой программы является обработка строк только в кодах ascii.c

2 Конструкторская часть

В данном разделе представлены схемы рассматриваемых алгоритмов и их оценка по памяти.

2.1 Схемы алгоритмов

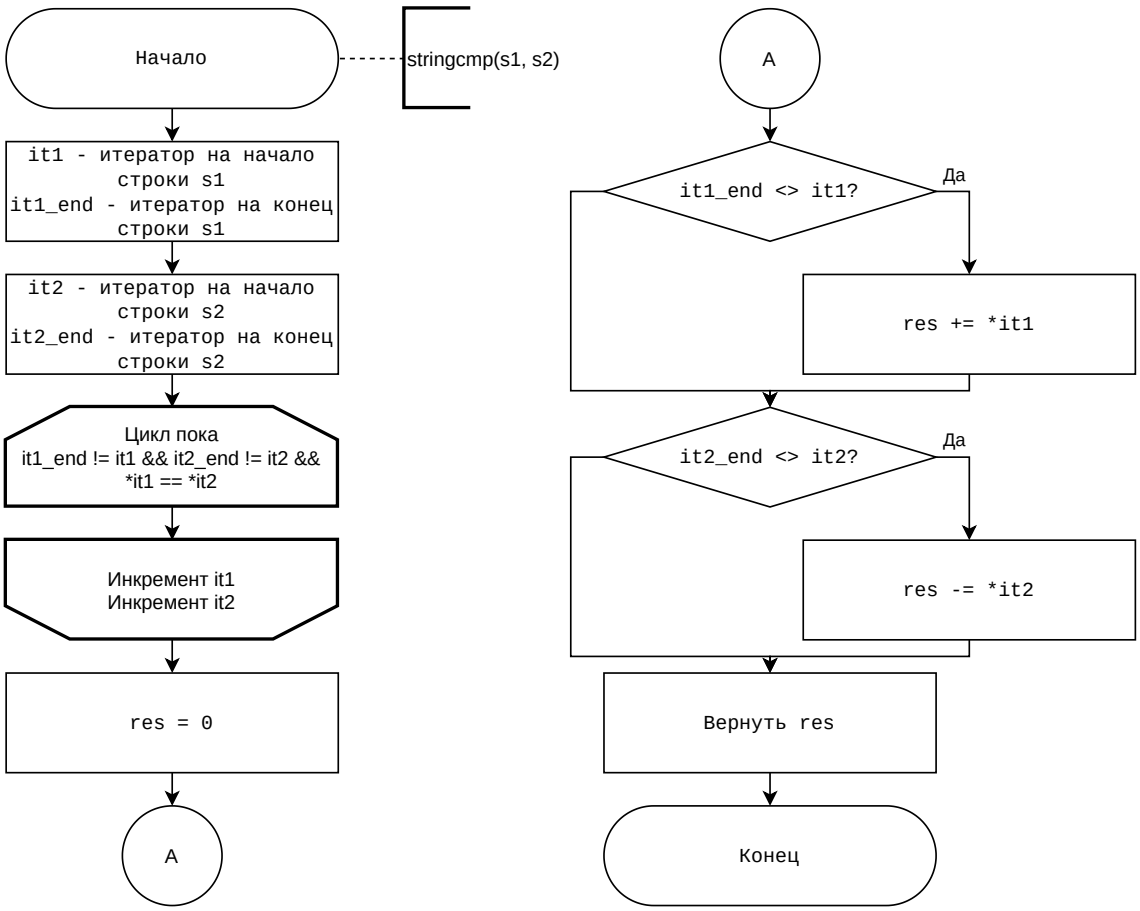

Рисунок 2.1 — Последовательный алгоритм лексикографического сравнения строк

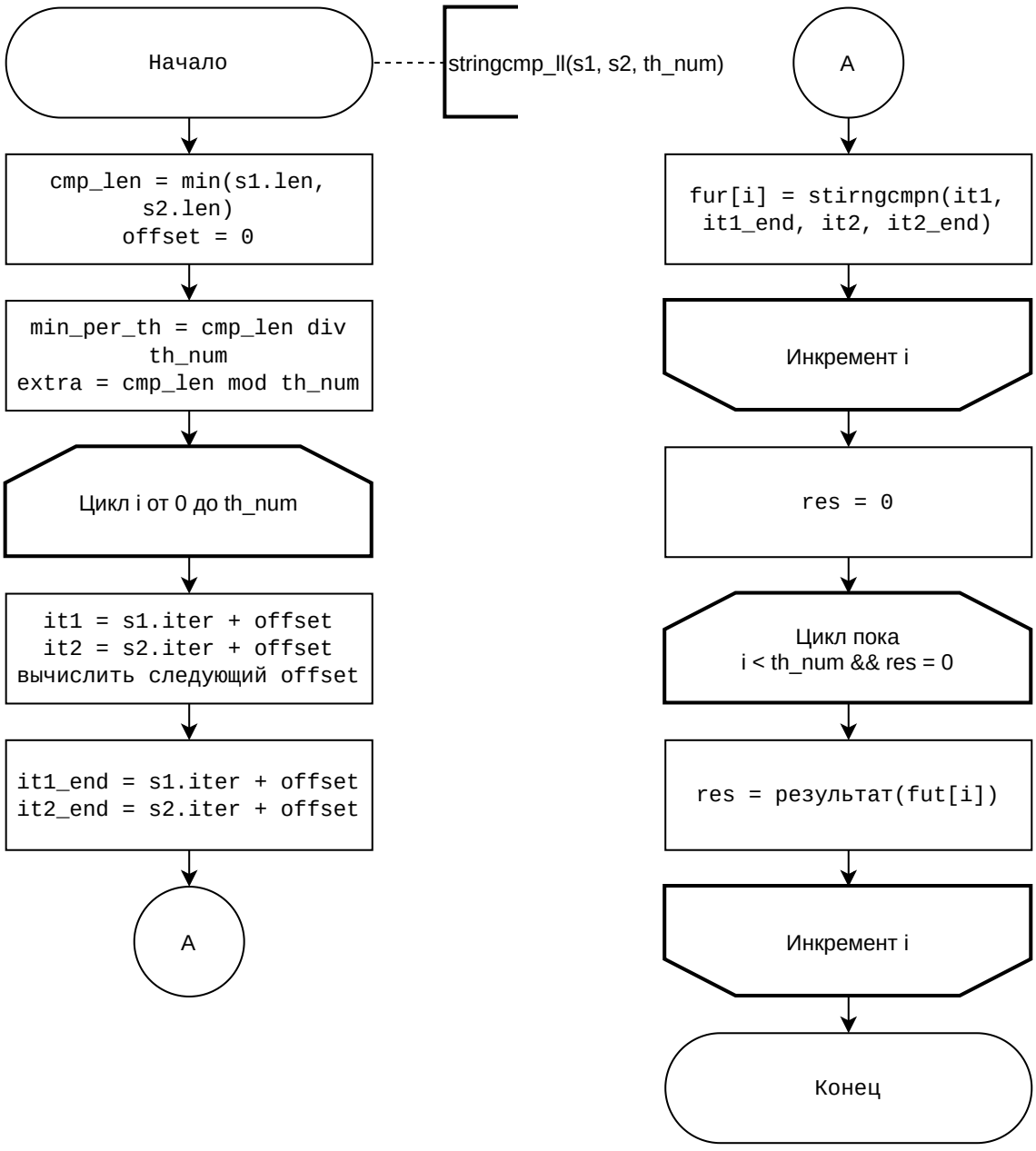

Рисунок 2.2 — Параллельный алгоритм лексикографического сравнения строк

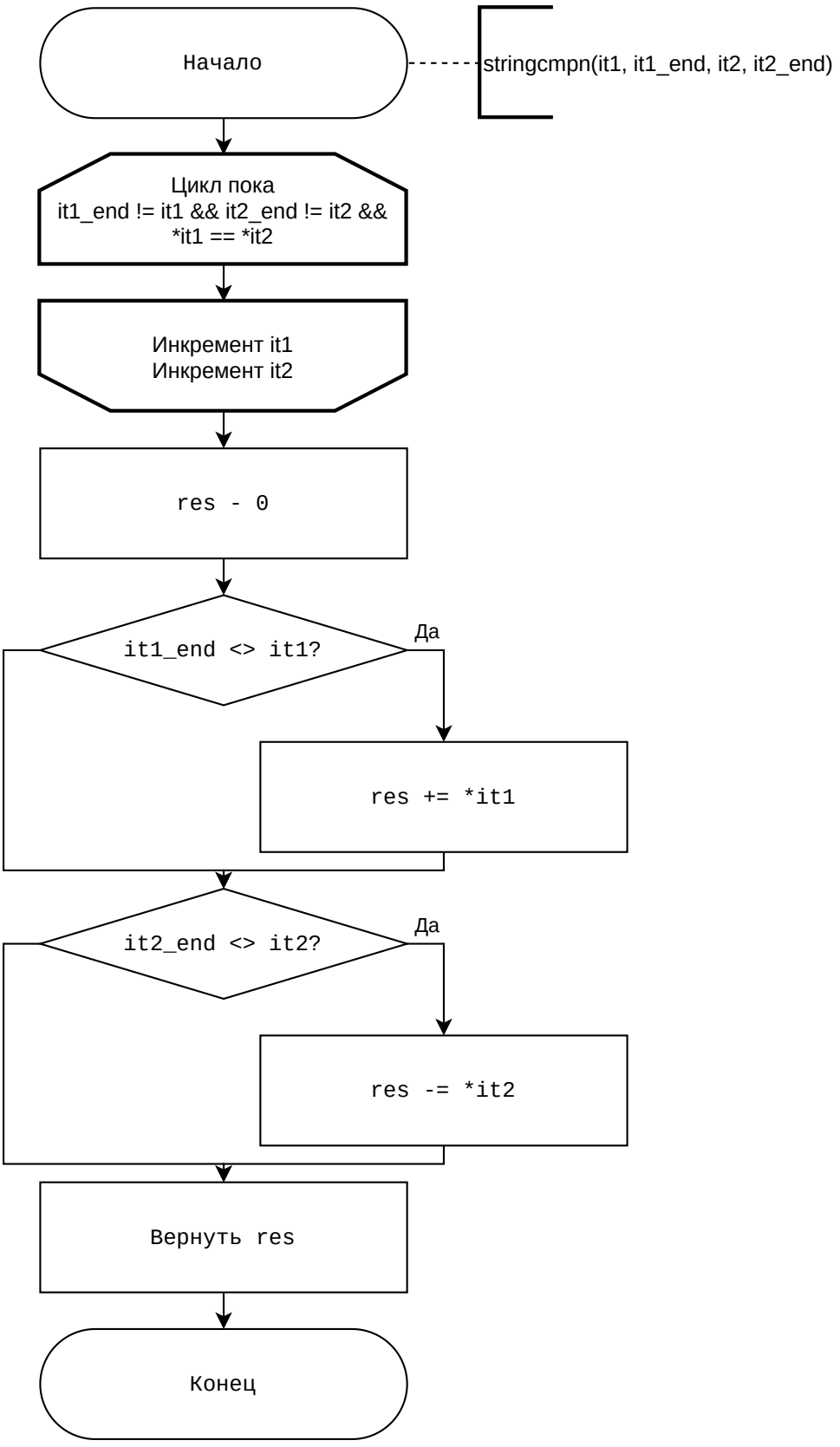

Рисунок 2.3 — Вспомогательная функция для параллельного алгоритма лексикографического сравнения строк

2.2 Структуры данных

В качестве входных данных алгоритмы принимают 2 строки, по которым рассчитывается искомое расстояние. Выходными данными алгоритма является число. Если число равно нулю, то строки равны, если число больше нуля, то первая строка больше второй иначе вторая строка больше первой.

В качестве строк для реализации алгоритма выберем нуль-терминировнаныее строки. Никаких других структур данных не потребуется.

Вывод

На основе теоретических данных, полученных из аналитического раздела были построены схемы требуемых алгоритмов.

3 Технологическая часть

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

3.1 Выбор средств реализации

Для реализации библиотеки с реализациями алгоритмов мною был выбрал язык C++[1] стандарта 2014 года. Он был выбран в силу предоставления библиотеки работы с физическими потоками.

Для юнит тестирования был выбран язык C++14, для использовая фрейворка GoogleTests[2].

3.2 Требования к программному обеспечению

К программе предъявляется ряд требований:

— На вход подаётся две регистрозависимые строки.

— На выходе — результат выполнения каждого из вышеуказанных алгоритмов.

3.3 Листинги кода

Листинг 3.1 — Основной файл программы main.cpp

```
#include <iostream>
#include <string>

#include "stringcmp/stringcmp.h"

int main() {
  std::string s1, s2;

  std::cout << "Enter 2 strings:" << std::endl;
  std::cin >> s1 >> s2;

  int res = stringcmp(s1, s2);
  int res_ll = stringcmp_ll(s1, s2);

  std::cout << "res          = " << res << std::endl;
  std::cout << "res parallel = " << res_ll << std::endl;
}
```

Листинг 3.2 — Последовательная реализация алгоритма

```
int stringcmp(std::string const& s1, std::string const& s2) {
  auto it1 = s1.cbegin();
  auto it1_end = s1.cend();

  auto it2 = s2.cbegin();
  auto it2_end = s2.cend();

  while (it1_end != it1 && it2_end != it2 && (*it1 == *it2)) {
    ++it1;
    ++it2;
  }

  return (it1_end == it1 ? 0 : *it1) - (it2_end == it2 ? 0 : *it2);
}
```

Листинг 3.3 — Параллельная реализация алгоритма

```
int stringcmp_ll(std::string const& s1, std::string const& s2,
                 const unsigned th_num) {
  std::vector<std::future<int>> futs(th_num);

  std::size_t cmp_len = std::min(s1.size(), s2.size());

  int offset = 0;
  unsigned min_per_th = cmp_len / th_num;
  unsigned extra = cmp_len % th_num;

  for (unsigned i = 0; i < th_num; ++i) {
    auto it1 = s1.cbegin() + offset;
    auto it2 = s2.cbegin() + offset;

    offset += min_per_th + (extra > 0 ? 1 : 0);

    auto it1_end = s1.cbegin() + offset;
    auto it2_end = s2.cbegin() + offset;

    --extra;

    futs[i] = std::async(
        std::launch::async,
        [](auto it1, auto it1_end, auto it2, auto it2_end) -> int {
          while (it1_end != it1 && it2_end != it2 && (*it1 == *it2)) {
            ++it1;
            ++it2;
          }

          return (it1_end == it1 ? 0 : *it1) - (it2_end == it2 ? 0 : *it2);
        },
        it1, it1_end, it2, it2_end);
  }

  int res = 0;
  for (unsigned i = 0; i < th_num && res == 0; res = futs[i].get(), ++i) {
  }

  return res;
}
```

Вывод

Были реализованы алгоритмы последовательного лексикографического сравнения строк и параллельного лексикографического сравнения строк.

4 Экспериментальная часть

В данном разделе будет проведено функциональное тестирование разработанного программного обеспечения. Так же будет произведено измерение временных характеристик и характеристик по памяти каждого из реализованных алгоритмов.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование:

— Процессор: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz [3]
— Оперативная память: 8 Gb
— Операционная система: Linux[4] elementary OS 6 Odin [5]

4.2 Тестирование

Юнит тестирование проводилось при помощи фрэймворка GoogleTests[2]. Были выполнены следующие тесты

Таблица 4.1 — Функциональные тесты

Строка 1	**Строка 2**	**Последовательный**	**Параллельный**
		0	0
Denis	Denis	0	0
abc	bc	1	1
bc	abc	-1	-1

При проведении функционального тестирования, полученные результаты работы программы совпали с ожидаемыми. Таким образом, функциональное тестирование пройдено успешно.

4.3 Временные характеристики

Для сравнения временных характеристик проведем сравнение времени работы параллельного алгоритма для разного числа потоков, а так же сравнения последовательного и параллельного алгоритма.

Данные эксперименты проведем для строк весом в 1, 2, 4, 8, 16, 32, 64 Мегабайта.

Результаты замеров времени при разном количестве потоков приведены в таблице 4.2.

Таблица 4.2 — Замер времени при разной длине строк и разном разном количестве потоков (время в миллисекундах)

Вес строк, Mb	**Количество потоков, ед**						
	1	**2**	**4**	**8**	**16**	**32**	**64**
1	21	9	4	3	4	4	4
2	33	18	14	8	7	7	8
4	66	33	16	15	14	14	15
8	132	66	34	29	33	30	31
16	265	136	77	61	65	59	59
32	536	267	151	130	127	120	118
64	1055	532	266	233	241	245	236
128	2117	1068	572	469	489	515	657
256	4255	2134	1070	945	1159	1118	1325
512	8475	4248	2751	2726	2707	2716	2695

Результаты замеров времени при выборе последовательного и параллельного алгоритма приведены в таблице 4.3.

На рисунке 4.1 приведен график сравнения последовательного и параллельного алгоритма лексикографического сравнения строк при строка разной длины. Параллельный алгоритм использовател 8 потоков. Параллельный алгоритм работает в среднем в 4-5 раз быстрее последовательного.

На рисунке 4.2 приведен график зависимости времени работы алгоритма от количества потоков для строки длины 256 Мегабайт. Наибольший выигрыш по времени происходит при количестве потоков равном 8, оно в 4.3 раза быстрее решения в 1 поток. При дальнейшем увеличении количества потоков время не уменьшается. На машине, на которой выполнялся данных эксперимент физически одновременно может выполнять 8 потоков. Следовательно, наибольший выигрыш по времени

наблюдается при запуске максимального поддерживаемого процессором количества потоков, которые могут выполняться одновременно.

Вывод

В данном разделе было произведено сравнение количества затраченного времени и памяти вышеизложенных алгоритмов.

Паралельный алгоритм лексикографиеческого сравнения строк для количестве потоков равном количеству ядер процессора в среднем работает быстрее последовательного в 4-5 раз. Наибольшим выигрыш наблюдается при числе потоков равном максимальному числу потоков, которые могут физически выполнять одновременно. Дальнейнее увеличение числа потоков к росту производительности не ведет.

Таблица 4.3 — Замер времени при разной длине строк и разных алгоритмах (время в миллисекундах)

Вес строк, Mb	**Параллельный**	**Непараллельный**
1	5	22
2	9	43
4	21	85
8	35	153
16	64	285
32	127	567
64	283	1126
128	570	2223
256	1111	4419
512	2342	8766

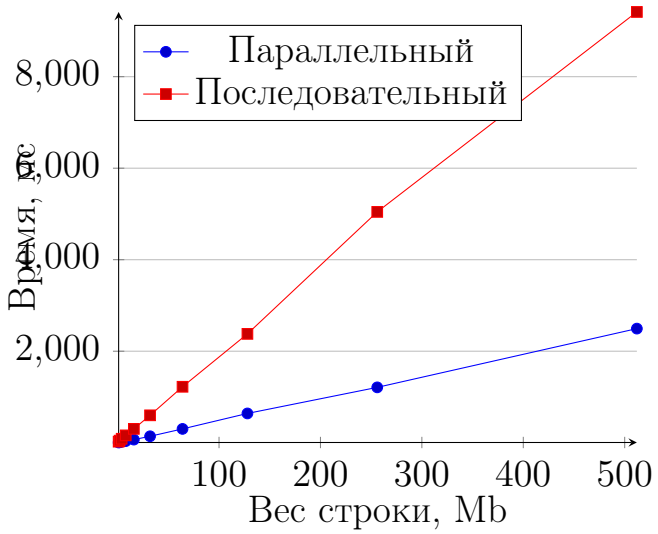

Рисунок 4.1 — Сравнение последовательного и параллельного алгоритма сравнения строк

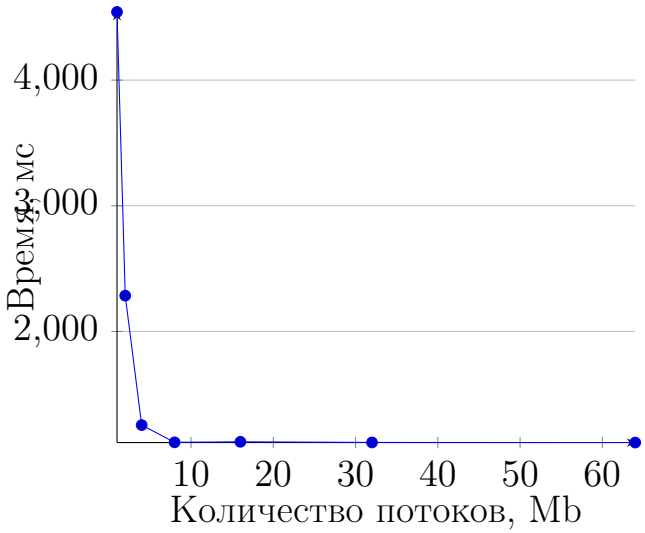

Рисунок 4.2 — Сравнение параллельного алгоритма при разном количестве потоков

Заключение

В результате выполнения данной лабораторной рабоыт были изучены алгоритмы лексикографического сравнения строк, построены схемы алгоритмов, реализованы данных алгоритмы и проведено их сравнение.

Паралельный алгоритм лексикографиеческого сравнения строк для количестве потоков равном количеству ядер процессора в среднем работает быстрее последовательного в 4-5 раз. Наибольшим выигрыш наблюдается при числе потоков равном максимальному числу потоков, которые могут физически выполнять одновременно. Дальнейнее увеличение числа потоков к росту производительности не ведет.

Список литературы

1. Стандарт C++ [Электронный ресурс]. — Режим доступа: https://www.iso.org/standard/64029.html (дата обращения: 30.10.2021).

2. Фреймворк Google tests. — Режим доступа: https://google.github.io/googletest/ (дата обращения: 11.10.2021).

3. Процессор Intel® Core™ i5-8250U. — Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/124967/ intel-core-i5-8250u-processor-6m-cache-up-to-3-40-ghz.html (дата обращения: 11.10.2021).

4. Ядро Linux. — Режим доступа: https://www.kernel.org/ (дата обращения: 30.10.2021).

5. Операциионная система elementary os. — Режим доступа: https://elementary.io/ (дата обращения: 11.10.2021).