**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 ИУ, Информатика и системы управления

КАФЕДРА ИУ7, Программное обеспечение ЭВМ и информационные технологии

# ЛАБОРАТОРНАЯ РАБОТА №4

## *ПО ДИСЦИПЛИНЕ*

## *“Анализ алгоритмов”*

Студент ИУ7-54Б _____ **А.А. Андреев**
(Группа) (Подпись, дата) (И.О.Фамилия)

Преподаватель _____ **Л.Л. Волкова**
(Подпись, дата) (И.О.Фамилия)

*2021 г.*

## Оглавление

# Введение.

Данная лабораторная работа посвящена изучению многопоточности.

**Многопоточность [1]** - способность центрального процессора (CPU) 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 (TLB).

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. Смысл многопоточности - квазимногозадачность на уровне одного исполняемого процесса. Значит, все потоки процесса помимо общего адресного пространства имеют и общие дескрипторы файлов. Выполняющийся процесс имеет как минимум один (главный) поток.

Многопоточность (как доктрину программирования) не следует путать ни с многозадачностью, ни с многопроцессорностью, несмотря на то, что операционные системы, реализующие многозадачность, как правило, реализуют и многопоточность.

Достоинства

- облегчение программы посредством использования общего адресного пространства.
- меньшие затраты на создание потока в сравнении с процессами.
- повышение производительности процесса за счёт распараллеливания процессорных вычислений.
- если поток часто теряет кэш, другие потоки могут продолжать использовать неиспользованные вычислительные ресурсы.

Недостатки

- несколько потоков могут вмешиваться друг в друга при совместном использовании аппаратных ресурсов

- с программной точки зрения аппаратная поддержка многопоточности более трудоемка для программного обеспечения
- проблема планирования потоков
- специфика использования. Вручную настроенные программы на ассемблере, использующие расширения MMX или AltiVec и выполняющие предварительные выборки данных, не страдают от потерь кэша или неиспользуемых вычислительных ресурсов. Таким образом, такие программы не выигрывают от аппаратной многопоточности и действительно могут видеть ухудшенную производительность из-за конкуренции за общие ресурсы.

**Цель данной лабораторной работы:** Изучение и реализация параллельных вычислений.

**Задачи данной лабораторной работы:**

1. Изучение понятие параллельных вычислений;
2. Реализовать последовательный и параллельные алгоритм Дамерау-Левенштейна;
3. Сравнить временные характеристики реализованных алгоритмов экспериментально;

# 1 Аналитическая часть

**Расстояние Левенштейна [1]** - это минимальное количество операций вставки/удаления/замены одного символа на другой, необходимых для превращения одной строки в другую. Каждая операция определяется своей ценой, в общем случае операции определены, как:

- $\omega(\lambda,\ b)$- цена операции вставки
- $\omega(a,\ \lambda)$- цена операции удаления
- $\omega(a,\ b)$- цена операции замены

Расстояние Левенштейна - это минимальная суммарная цена после последовательности замен. Существуют частные случаи нахождения расстояния Левенштейна:

- $\omega(a,\ a)\ =\ 0$
- $\omega(\lambda,\ b)\ =\ 1$
- $\omega(a,\ \lambda)\ =\ 1$
- $\omega(a,\ b)\ =\ 1$ и $a\ \neq\ b$

## 1.1 Алгоритм Дамерау-Левенштейна [3]

Формула (1.3) нахождения расстояния Дамерау-Левенштейна определяется так, как и (1.1) - формула неэффективна по времени и аналогично для оптимизации используется добавление матрицы для хранения промежуточных значений.

$$d_{a,b}(i,j) = \begin{cases} \max(i,j), & \text{если } \min(i,j) = 0, \\ \min\{ & \\ \quad d_{a,b}(i,j-1)+1, & \\ \quad d_{a,b}(i-1,j)+1, & \\ \quad d_{a,b}(i-1,j-1)+m(a[i],b[j]), & \text{иначе} \\ \quad \begin{cases} d_{a,b}(i-2,j-2)+1, & \text{если } i,j>1; \\ & a[i]=b[j-1]; \\ & b[j]=a[i-1] \\ \infty, & \text{иначе} \end{cases} & \\ \} & \end{cases}, \qquad (1.3)$$

## 1.2 Выводы из аналитического раздела

В данном разделе был описан итерационный алгоритм Дамерау-Левенштейна, который показывает, что мы имеет возможность разбить на потоки независимые друг от друга вычисления смена позиций рядов, подсчет стоимости операций.

# 2 Конструкторская часть.

В данном разделе будет приведены блок-схемы алгоритмов, описанных в аналитическом разделе п.1.

## 2.1 Схемы алгоритмов

### Итеративный алгоритм Дамерау-Левенштейна

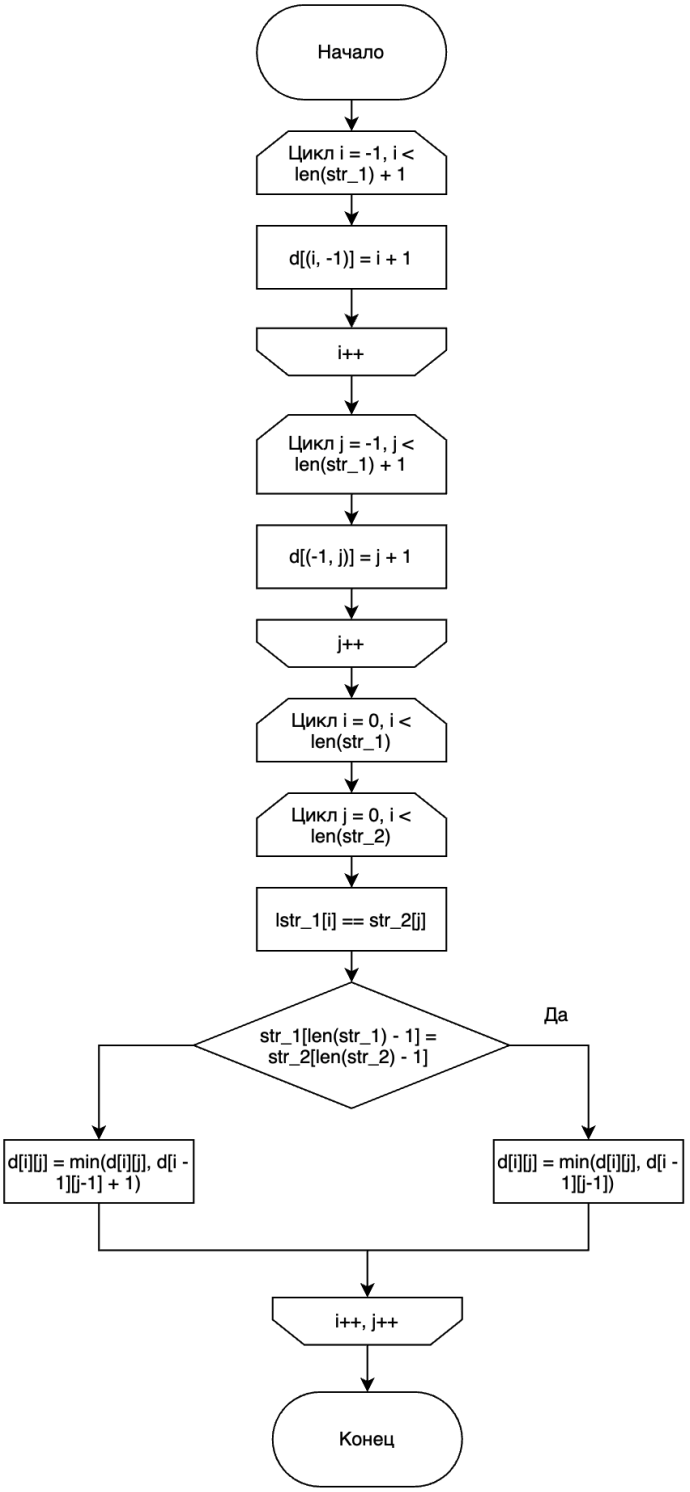


Рисунок 1: Схема итеративного алгоритма Дамерау-Левенштейна

**Итеративный алгоритм Дамерау-Левенштейна с распараллеливанием**

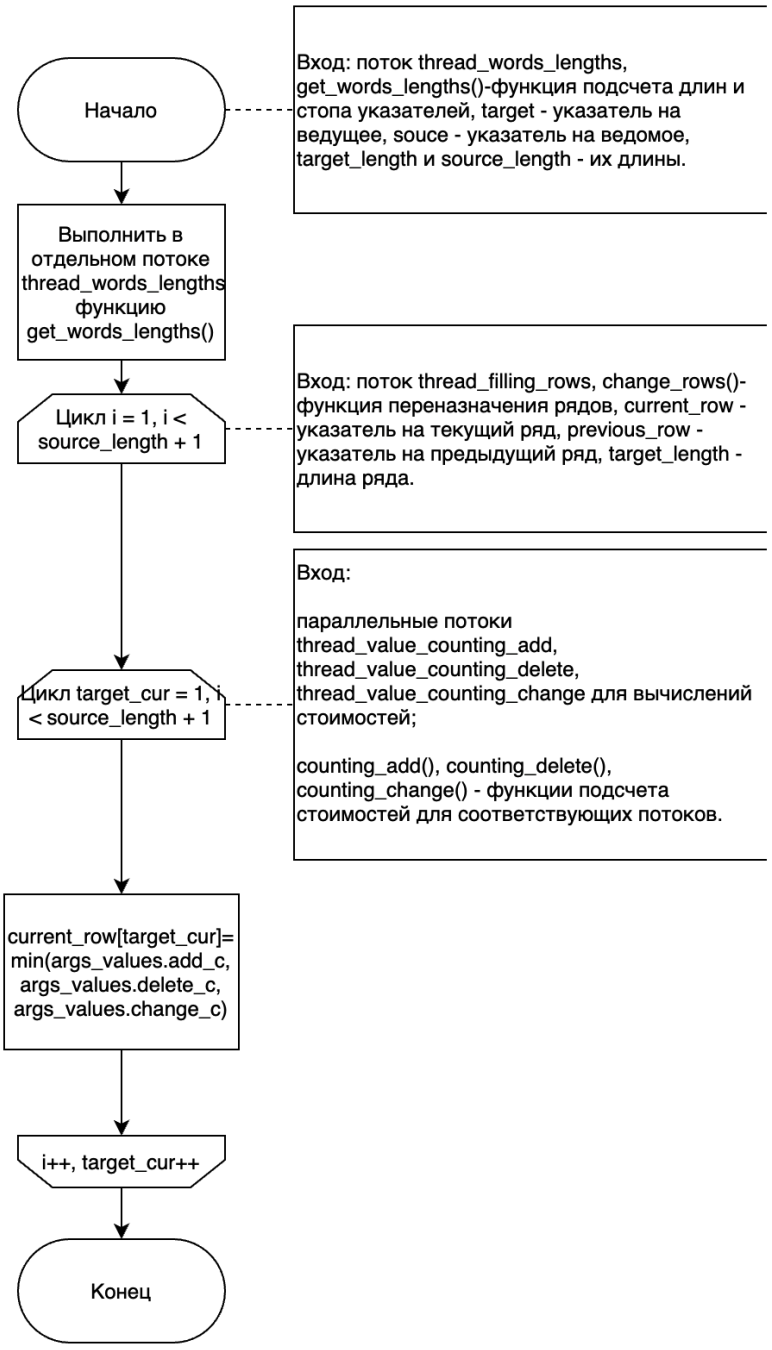


Рисунок 2: Схема итеративного алгоритма Дамерау-Левенштейна с распараллеливанием

## 2.2 Вывод

Блок-схемы в данном разделе демонстрируют схемы работы итеративного алгоритм Дамерау-Левенштейна в классической версии и с распараллеливанием.

# 3 Технологическая часть.

В данном разделе будут рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач.

## 3.1 Требования к программному обеспечению

Программное обеспечение должно реализовывать поставленную на лабораторную работу задачу. Интерфейс для взаимодействия с программой - командная строка. Программа должна выводить полученное расстояние между двумя введенными строками и показывать потраченное на это время.

## 3.2 Выбор и обоснование языка и среды программирования.

Для разработки данной программы применён язык Си с библиотекой getCPUTime.c [4] для вычисления времени работы процессора, чтобы расширить знания в области данного языка программирования.

## 3.3 Реализация алгоритмов

В листингах 1-5 приведена реализация алгоритмов нахождения расстояния Дамерау-Левенштейна классическим способом и при помощи разделения на потоки.

Листинг 1: Функция нахождения расстояния Дамерау-Левенштейна, Часть 1

```
1. // Получение расстояния между словами без распараллеливания
2. void out_levenstein_distance(char *target, char *source)
3. {
4.       clock_t begin = clock();
5.       // Инициализация основного поток вычисления и обработки
   длин
6.       // pthread_t thread_words_lengths;
7.
8.       // Результирующая длина
9.       int result_levenstein_distance = 0;
10.
11.      // Длины слов
12.      int target_length = 0, source_length = 0;
13.
14.      // Обновление длин и смена местами
15.      update_words_length(&target, &target_length, &source,
   &source_length);
16.
17.      // Выделение памяти под текущую строку
18.      int *current_row = (int*) malloc(MAX_WORD_SIZE);
19.
20.      for (int check_cur = 0; check_cur < target_length + 1;
   check_cur++)
21.           current_row[check_cur] = check_cur;
22.
```

Листинг 2: Функция нахождения расстояния Дамерау-Левенштейна, Часть 2

```
23.      for (int i = 1; i < source_length + 1; i++)
24.      {
25.            // Выделение памяти
26.            int *previous_row = (int *) malloc(MAX_WORD_SIZE);
27.
28.            // Переназначение рядов
29.            for (int check_cur = 0; check_cur < target_length +
   1; check_cur++)
30.            {
31.                  previous_row[check_cur] =
   current_row[check_cur];
32.
33.                  if (check_cur == 0)
34.                        current_row[check_cur] = i;
35.                  else
36.                        current_row[check_cur] = 0;
37.            }
38.
39.            // Подсчет
40.            for (int target_cur = 1; target_cur < target_length
   + 1; target_cur++)
41.            {
42.                  int add_c = previous_row[target_cur] + 1;
43.                  int delete_c = current_row[target_cur - 1];
44.                  int change_c = previous_row[target_cur - 1];
45.
46.                  if (target[target_cur - 1] != source[i - 1])
47.                        change_c += 1;
48.
49.                              current_row[target_cur] =
   min(min(add_c, delete_c), change_c);
50.                        }
51.                  }
52.
53.
54.      /* here, do your time-consuming job */
55.
56.      clock_t end = clock();
57.      result_levenstein_distance = current_row[target_length];
58.
59.      printf("\nКлассическая реализация:\n");
60.      printf("\tВремя: %f мкс\n", (double)(end - begin) /
   CLOCKS_PER_SEC * 1000);
61.
   out_levestein_get_distance_result(result_levenstein_distance);
62.   }
```

Листинг 3: Функция нахождения расстояния Дамерау-Левенштейна при помощи реализации разделения на потоки, Часть 1

```
1. // Получение расстояния между словами с распараллеливанием
2. void out_levenstein_distance_parall(char *target, char *source)
3. {
4.       clock_t begin = clock();
5.       // Инициализация основного поток вычисления и обработки
   длин
6.       pthread_t thread_words_lengths;
7.
8.       // Результирующая длина
9.       int result_levenstein_distance = 0;
10.
11.      // Длины слов
12.      int target_length = 0, source_length = 0;
13.
14.      Args_words args_words_lengths;
15.      args_words_lengths.target = target;
16.      args_words_lengths.source = source;
17.      int get_words_length_status =
   pthread_create(&thread_words_lengths, NULL, get_words_lengths,
   (void*) &args_words_lengths);
18.
19.      // Ожидание завершения потока
20.      pthread_join(thread_words_lengths, NULL);
21.
22.      // Возврат значений из обработки потока
23.      target = args_words_lengths.target;
24.      source = args_words_lengths.source;
25.      target_length = args_words_lengths.target_length;
26.      source_length = args_words_lengths.source_length;
27.
28.      if (get_words_length_status != SUCCESS_STATUS)
29.            out_error_message();
30.
31.      // Выделение памяти под текущую строку
32.      int *current_row = (int*) malloc(MAX_WORD_SIZE);
33.
34.      for (int check_cur = 0; check_cur < target_length + 1;
   check_cur++)
35.            current_row[check_cur] = check_cur;
36.
37.      for (int i = 1; i < source_length + 1; i++)
38.      {
39.            // Инициализация поток заполнения ряда
40.            pthread_t thread_filling_rows;
41.
42.            // Выделение памяти
43.            int *previous_row = (int *) malloc(MAX_WORD_SIZE);
44.
45.            Args_rows args_rows;
46.
```

Листинг 4: Функция нахождения расстояния Дамерау-Левенштейна при помощи реализации разделения на потоки, Часть 2

```
47.   args_rows.current_row = current_row;
48.           args_rows.previous_row = previous_row;
49.           args_rows.i = i;
50.           args_rows.target_length = target_length;
51.
52.           int change_rows_status =
   pthread_create(&thread_filling_rows, NULL, change_rows, (void*)
   &args_rows);
53.
54.           // Установка данных из потока
55.           current_row = args_rows.current_row;
56.           previous_row = args_rows.previous_row;
57.
58.           if (change_rows_status != SUCCESS_STATUS)
59.                 out_error_message();
60.
61.           for (int target_cur = 1; target_cur < target_length
   + 1; target_cur++)
62.           {
63.                 // Инициализация структуры
64.                 Args_values args_values;
65.
66.                 // Передача значений в структуру
67.                 args_values.current_row = current_row;
68.                 args_values.previous_row = previous_row;
69.                 args_values.target_cur = target_cur;
70.                 args_values.source = source;
71.                 args_values.target = target;
72.                 args_values.source_cur = i;
73.
74.                 // Создание группы параллельных потоков
   вычислений стоимости
75.                 pthread_t thread_value_counting_add;
76.                 pthread_t thread_value_counting_delete;
77.                 pthread_t thread_value_counting_change;
78.
79.                 // Переназначение
80.                 int counting_add_status =
   pthread_create(&thread_value_counting_add, NULL, counting_add,
   (void*) &args_values);
81.                 int counting_detele_status =
   pthread_create(&thread_value_counting_delete, NULL,
   counting_delete, (void*) &args_values);
82.                 int counting_change_status =
   pthread_create(&thread_value_counting_change, NULL,
   counting_change, (void*) &args_values);
83.
```

Листинг 5: Функция нахождения расстояния Дамерау-Левенштейна при помощи реализации разделения на потоки, Часть 3

```
84.
                    // Ожидание завершения потока
85.                 pthread_join(thread_value_counting_add, NULL);
86.                 pthread_join(thread_value_counting_delete,
    NULL);
87.                 pthread_join(thread_value_counting_change,
    NULL);
88.
89.                 if (counting_add_status != SUCCESS_STATUS ||
90.                       counting_detele_status != SUCCESS_STATUS
    ||
91.                       counting_change_status != SUCCESS_STATUS)
92.                       out_error_message();
93.
94.                 current_row[target_cur] =
    min(min(args_values.add_c, args_values.delete_c),
    args_values.change_c);
95.           }
96.      }
97.
98.      clock_t end = clock();
99.      result_levenstein_distance = current_row[target_length];
100.
101.     printf("\nМногопоточная реализация:\n");
102.     printf("\tВремя: %f мкс\n", (double)(end - begin) /
    CLOCKS_PER_SEC * 1000);
103.
    out_levestein_get_distance_result(result_levenstein_distance);
104.  }
```

### 3.4 Тестовые данные

Тестовые данные, на которых было протестировано разработанное программное обеспечение, представлено в Таблице 1.

Таблица 1: Тестовые данные, Часть 1

| № | Первое слово | Второе слово | Ожидаемый результат | | Полученный результат | |
|---|---|---|---|---|---|---|
| | | | Расстояние | | Расстояние | |
| | | | Обычный | Потоки | Обычный | Потоки |
| 1 | увлечение | развлечения | 4 | 4 | 4 | 4 |
| 2 | кот | скат | 2 | 2 | 2 | 2 |
| 3 | “” | тест | 4 | 4 | 4 | 4 |
| 4 | мгту | мтгу | 2 | 2 | 2 | 2 |

Таблица 2: Тестовые данные, Часть 2

| № | Первое слово | Второе слово | Ожидаемый результат | | Полученный результат | |
|---|---|---|---|---|---|---|
| | | | Расстояние | | Расстояние | |
| | | | Обычный | Потоки | Обычный | Потоки |
| 5 | рот | кот | 1 | 1 | 1 | 1 |
| 7 | лилия | рим | 4 | 4 | 4 | 4 |
| 8 | рим | мир | 2 | 2 | 2 | 2 |
| 9 | apple | aplpe | 2 | 2 | 2 | 2 |
| 10 | катя | надя | 2 | 2 | 2 | 2 |

### 3.5 Вывод

В данном разделе были рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач, приведены результаты работы программы на тестовых данных.

# 4 Исследовательская часть.

## 4.1 Демонстрация работы программы

Пример работы программы представлен на рисунке 3.

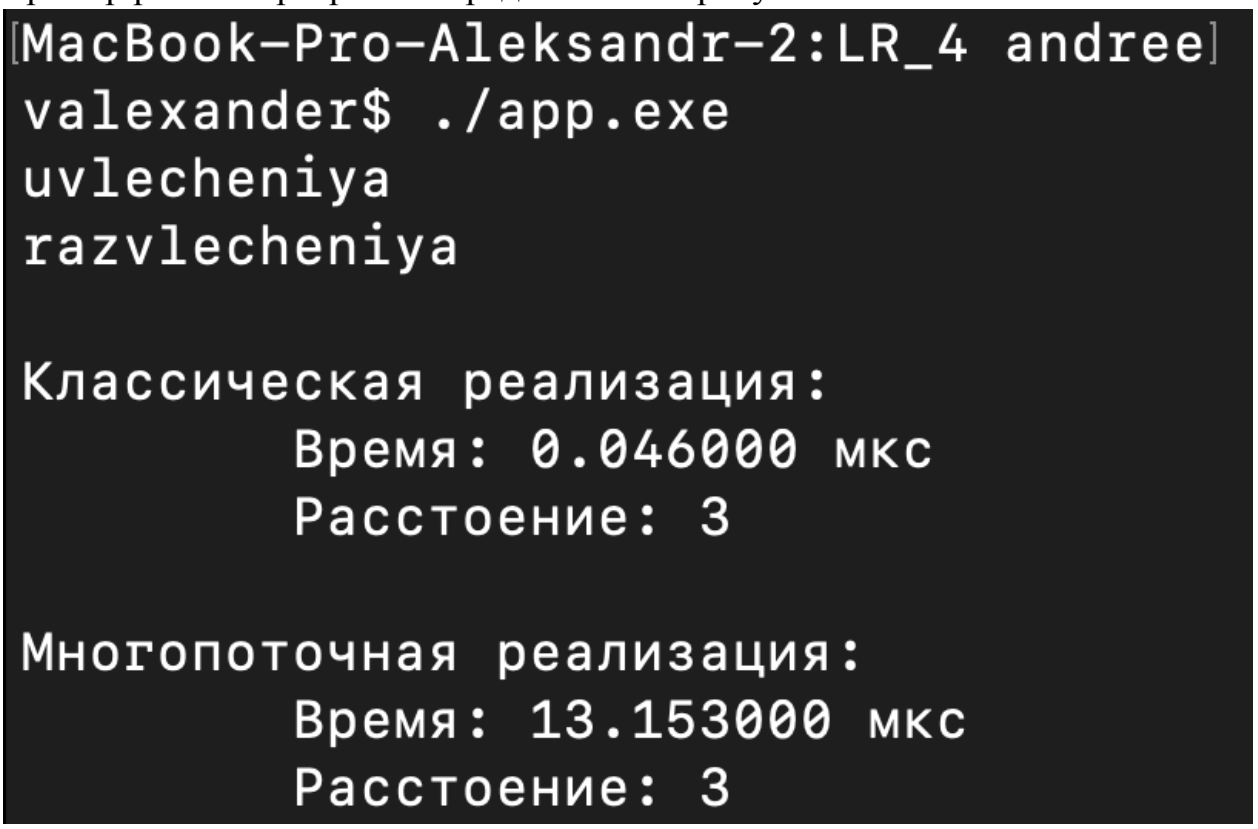


Рисунок 3: Демонстрация работы программы на примере строк Uvlecheniya и Razvlecheniya

## 4.2 Технические характеристики

В Таблице 3. приведены технические характеристики ЭВМ, на котором проводилось тестирование разрабатываемого программного обеспечения.

Таблица 3: Технические характеристики ЭВМ, на котором проводилось тестирование разрабатываемого программного обеспечения

| | |
|---|---|
| ОС | Mac OS Mojave 64-bit |
| ОЗУ | 8 Gb 2133 MHz LPDDR3 |
| Процессор | 2,3 GHz Intel Core i5 |

## 4.3 Время выполнения алгоритмов

В Таблице 4. приведена информация о времени выполнения алгоритма Дамерау-Левенштейна с различным количеством потоков в микросекундах.

Таблица 4: Таблица времени выполнения алгоритма Дамерау-Левенштейна с различным количеством потоков (в микросекундах)

| № | Длина строк | Время | |
|---|---|---|---|
| | | Обычный | П1 |
| 1 | 10 | 1,28 | 0,49 |
| 2 | 20 | 6,58 | 1,84 |
| 3 | 40 | 39,35 | 10,96 |
| 4 | 80 | 302,11 | 83,95 |
| 5 | 160 | 2968,58 | 816,41 |

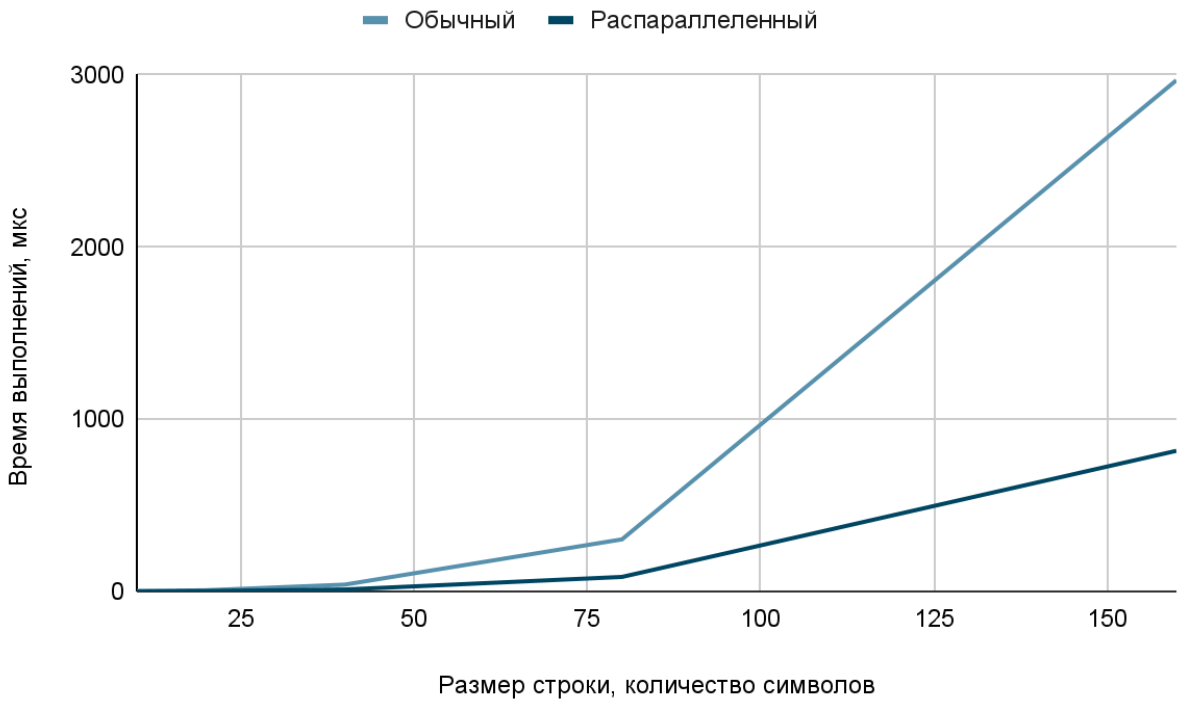


Рисунок 4: График времени выполнения алгоритмов на случайных данных с четной размерностью (в микросекундах) на основе Таблицы 4

### 4.4 Вывод

Наилучшее время параллельные алгоритмы показали на 4 потоках, что соответствует количеству логических ядер компьютера, на котором проводилось тестирование. На матрицах размером 512 на 512, параллельные алгоритмы улучшают время обычной (однопоточной) реализации поиска расстояния Левенштейна примерно в 3.5 раза. При количестве потоков, большее чем 4, параллельная реализация замедляет выполнение (в сравнении с 4 потоками).

# Заключение.

В рамках данной лабораторной работы была достигнута ее цель: изучены параллельные вычисления, также выполнены следующие задачи:

- было изучено понятие параллельных вычислений;
- были реализованы обычный и 2 параллельных реализаций алгоритма нахождения расстояния Дамерау-Левенштейна;
- было произведено сравнение временных характеристик реализованных алгоритмов экспериментально.

Параллельные реализации алгоритмов выигрывают по скорости у обычной (однопоточной) реализации. Наиболее эффективны данные алгоритмы при количестве потоков, совпадающем с количеством логических ядер компьютера. Так, например, на строке размером 160, удалось улучшить время выполнения алгоритма умножения матриц в 3.6 раза (в сравнении с однопоточной реализацией).

# Список использованной литературы

[1] Jesse Russel и Ronald Cohn, Расстояние Левенштейна, год выпуска 2010, тираж 3800, 690 страницы.

[2] Потоки в Си [Электронный ресурс] Режим доступа: https://younglinux.info/cthreat/inheritance. Дата обращения: 13.09.2021

[3] Гасфилд, Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология. Невский Диалект БВХ-Петербург, год выпуска 2003, типаж 900, 653 страницы.

[4] Вычисление процессорного времени выполенения программы [Электронный ресурс] Режим доступа: https://www.gnu.org/software/libc/manual/html_node/CPU-Time.html Дата обращения: 13.09.2021