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 дисциплине «Анализ алгоритмов»

Тема Параллельные вычисления на основе нативных потоков

Студент Шматко К. М.

Группа ИУ7-56Б

Оценка (баллы) _____

Преподаватель: Волкова Л. Л.

Москва — 2023 г.

Содержание

Введение

С появлением более совершенных вычислительных систем программистам стало необходимо выполнять одновременную обработку данных для улучшения реакции системы, увеличения скорости вычислений и более разумного использования вычислительных ресурсов. Метод многопоточности позволяет процессам и потокам совместно использовать ресурсы ядер, такие как вычислительные блоки, кэш-память и буфер перевода с преобразованием.

Многопоточность фактически представляет собой возможность выполнять несколько задач в пределах одного процесса, где потоки в процессе имеют общие адресные пространства и дескрипторы файлов. Каждый выполняемый процесс имеет по крайней мере один главный поток.

Целью данной лабораторной работы является исследование параллельных вычислений на материале работы с библиотекой *pymorphy*2.

Для достижения поставленной цели необходимо выполнить следующие задачи.

1) Описать понятие параллельных вычислений и работу с библиотекой *pymorphy*2.

2) Реализовать последовательный и параллельный алгоритм работы.

3) Замерить время реализации.

4) Провести анализ затрат работы программы по времени.

1 Аналитическая часть

В этом разделе будут представлена информация о многопоточности и исследуемом алгоритме получения гипотез для неоднозначных словоформ.

1.1 Многопоточность

Многопоточность – способность центрального процессора обеспечивать параллельное выполнение нескольких потоков в рамках использования ресурсов одного процессора [1]. Поток – это последовательность инструкций, которые могут выполняться параллельно с другими потоками внутри того же процесса, из которого они вытекают.

Процесс – это программа в процессе выполнения [2]. При запуске программы или приложения создается процесс, который может включать в себя один или несколько потоков. Потоки представляют собой сегменты процесса, осуществляющие выполнение задач, поставленных перед самим приложением. Процесс завершается, когда все его потоки завершают свою работу. Каждый поток в операционной системе представляет собой задачу, которую должен выполнить процессор. В настоящее время большинство процессоров способны обрабатывать несколько задач на одном ядре путем создания виртуальных ядер или обладают несколькими физическими ядрами. Такие процессоры называются многоядерными.

Одной из проблем, встающих при использовании потоков, является проблема совместного доступа к информации. Фундаментальным ограничением является запрет на запись из двух и более потоков в одну ячейку памяти одновременно.

1.2 Библиотека *pymorphy2*

Библиотека *pymorphy2* – морфологический анализатор, написанный на языке Python [3]. Он умеет следующее.

1) Приводить слово к нормальной форме (например, «люди – человек», или «гулял – гулять»).

2) Ставить слово в нужную форму. Например, ставить слово во множественное число, менять падеж слова и т.д.

3) Возвращать грамматическую информацию о слове (число, род, падеж, часть речи и т.д.).

В *pymorphy2* для морфологического анализа слов есть класс $MorphAnalyzer$. Метод $MorphAnalyzer.parse()$ возвращает один или несколько объектов типа $Parse$ с информацией о том, как слово может быть разобрано. Для получения формологических свойств словоформы используется тег – набор граммем, характеризующих данное слово.

Словоформы для которых возвращается несколько вариантов гипотез являются неоднозначными.

Для реализации параллельного алгоритма необходимо разбить входной файл с неоднозначными словоформами на n равных частей, где n – количество потоков, и определить слова в n массивов. Так как каждый поток будет работать со своим входным массивом словоформ, то нет необходимости в введении средств синхронизации: мьютексов, семафоров.

Вывод

В данном разделе была представлена информация о многопоточности и исследуемом алгоритме.

2 Конструкторская часть

В данном разделе будут приведены схемы алгоритма получения гипотез для неоднозначных словоформ в последовательной и параллельной реализации.

2.1 Разработка алгоритмов

На рисунке 2.1 представлена схема алгоритма получения гипотез для неоднозначных словоформ. На рисунке 2.2 – схема для запуска алгоритма без вспомогательных потоков. На рисунке 2.3 – схема для запуска алгоритма с вспомогательными потоками.

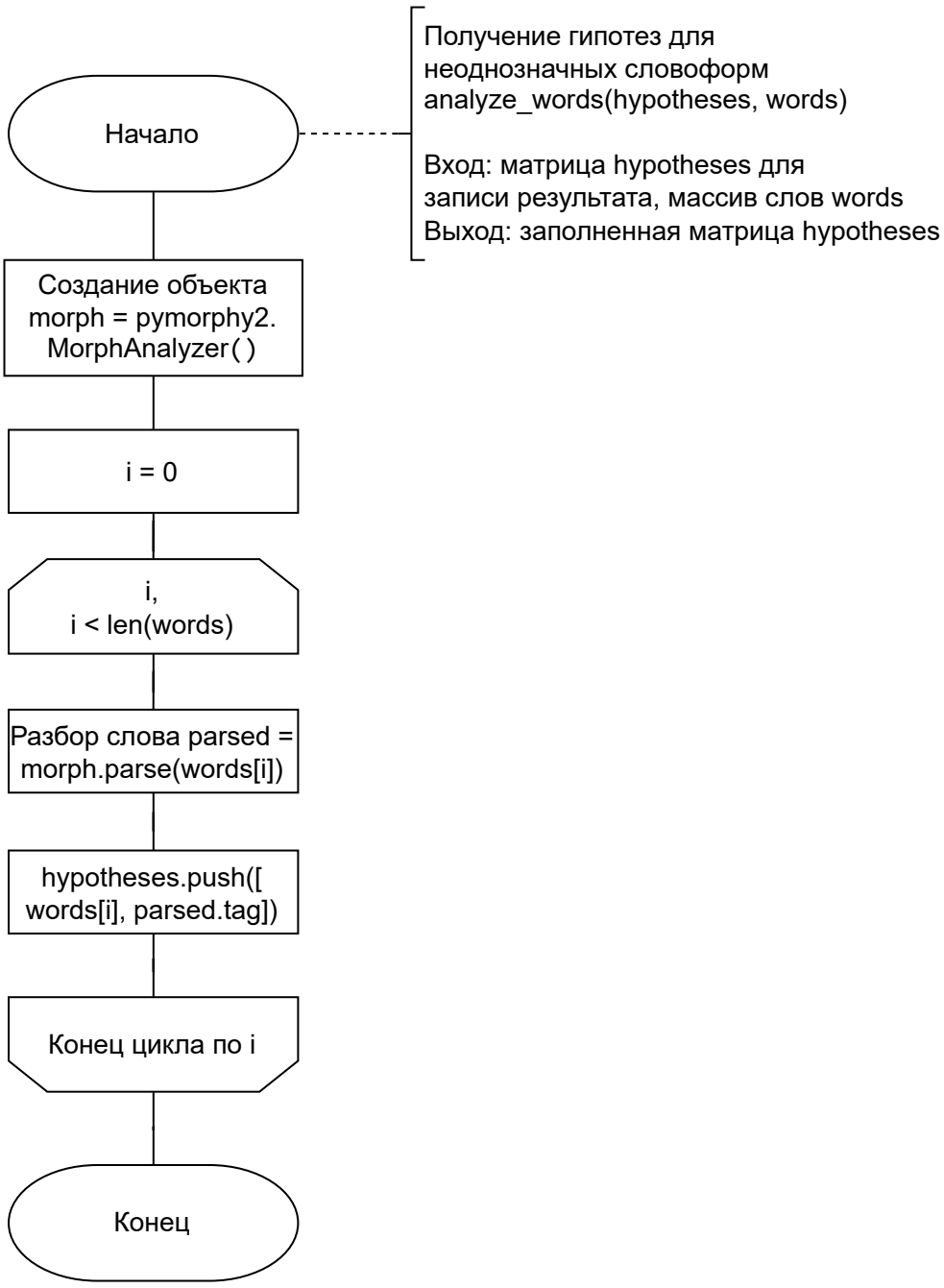

Рисунок 2.1 – Схема алгоритма алгоритма получения гипотез для неоднозначных словоформ

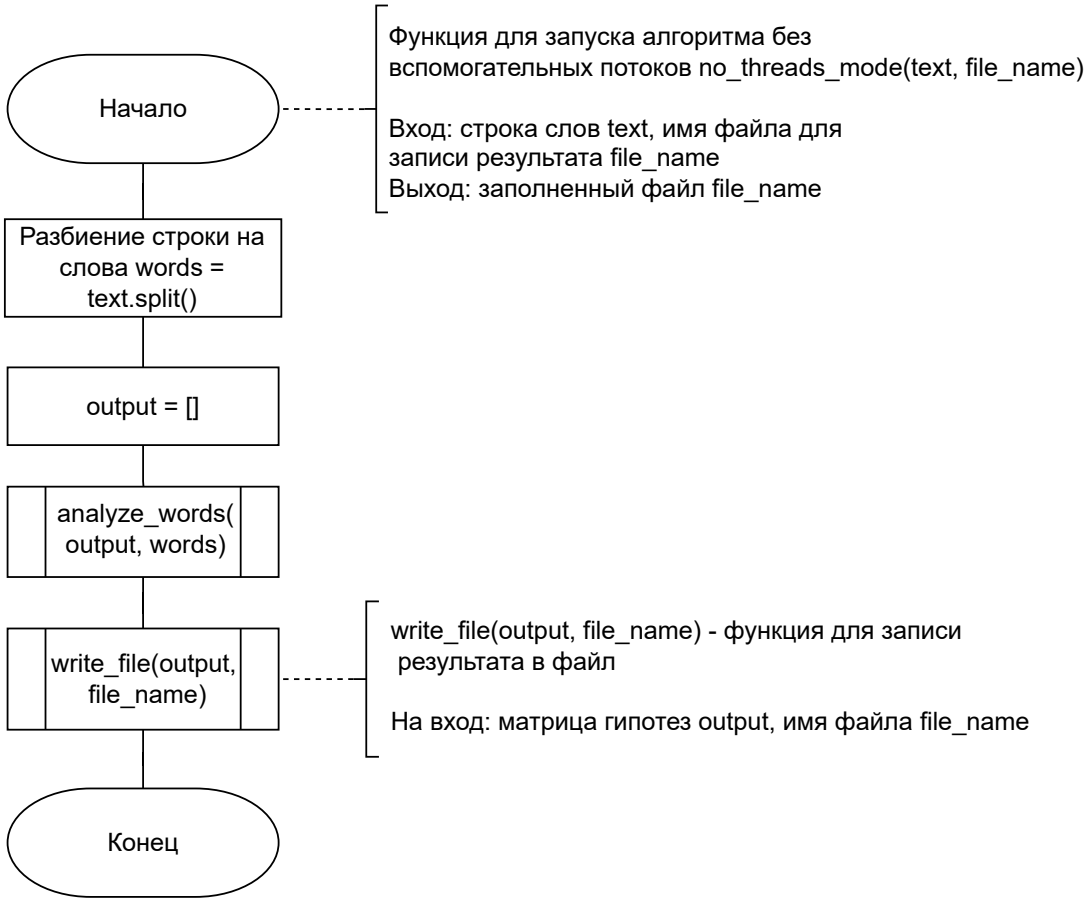

Рисунок 2.2 – Схема функции для запуска алгоритма без вспомогательных потоков

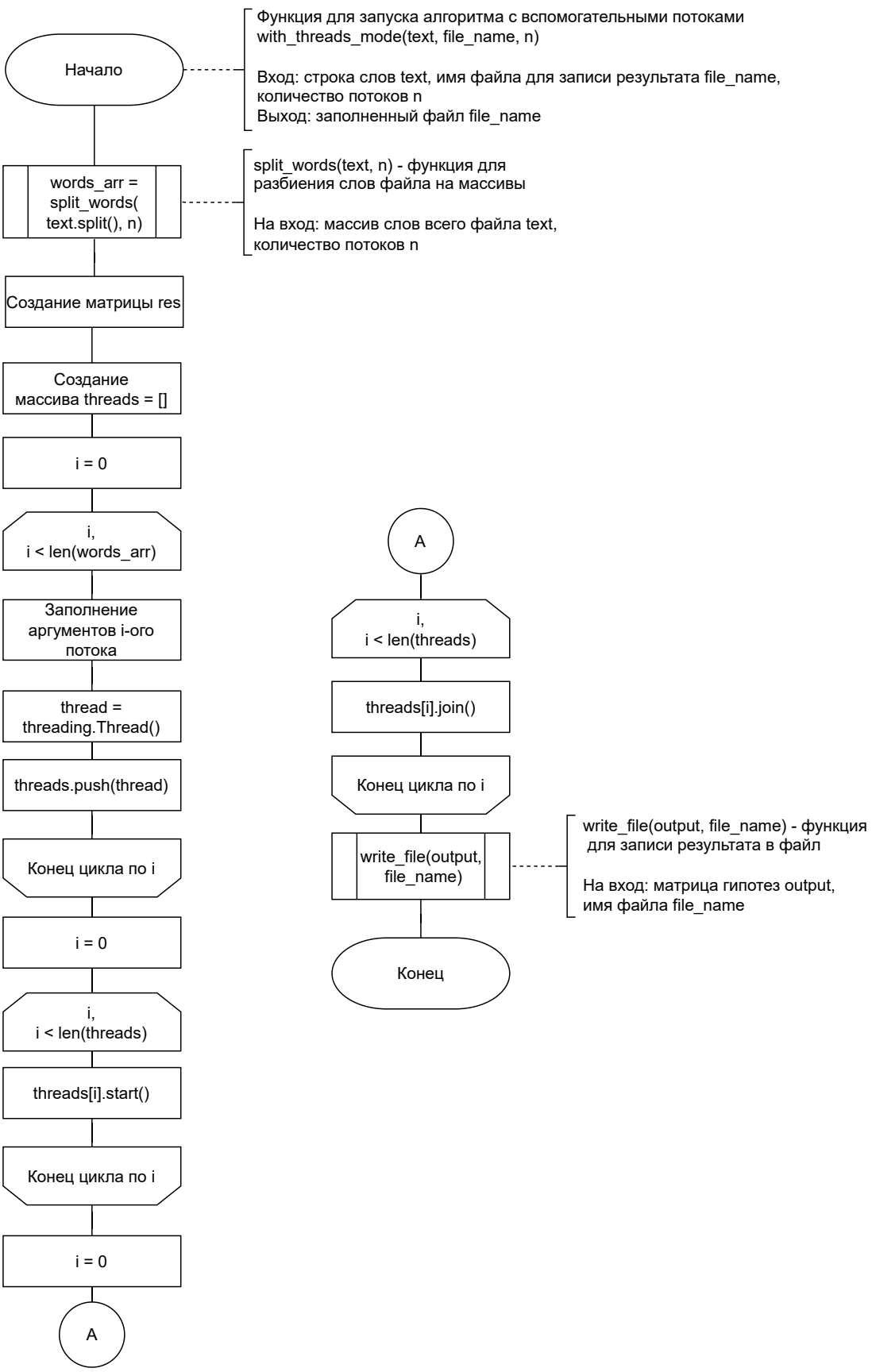

Рисунок 2.3 – Схема функции для запуска алгоритма с вспомогательными потоками

Вывод

В данном разделе на основе теоретических данных были построены схемы требуемых алгоритмов.

3 Технологическая часть

В данном разделе будут описаны требования к программному обеспечению, средства реализации, листинг кода и функциональные тесты.

3.1 Требования к программному обеспечению

К программе предъявлен ряд требований:

— принимает на вход файл с неоднозначными словоформами;

— создает и записывает в файл результат работы алгоритма;

— имеет интерфейс для выбора действий;

— имеет функциональность замера времени работы реализаций алгоритма.

3.2 Средства реализации

Для реализации данной лабораторной работы был выбран язык Python [4]. Данный выбор обусловлен наличием у языка всех требующихся инструментов для данной лабораторной работы, а также возможностью работать с библиотекой *pymorphy2*.

Время выполнения программы было замерено с использованием функции *time_ns*() из библиотеки *time* [5].

Для работы с потоками использовались функции библиотеки *threading* [6]. Чтобы работать с сущностью вспомогательного потока необходимо воспользоваться функцией *threading.Thread*() для создания потока и указания функции, которую будет выполнять созданный поток. Для запуска потока необходимо вызвать метод *start*(). Далее при помощи вызова *join*() необходимо дождаться завершения всех вспомогательных потоков, чтобы в главном потоке обработать результаты их работы.

3.3 Сведения о модулях программы

Данная программа разбита на следующие модули:

— `main.py` — файл, содержащий точку входа в программу, из которой происходит запуск работы реализаций алгоритма.

— `file.py` — файл, содержащий функции для работы с файлами.

— `graph.py` — файл, содержащий функции для построения графиков.

3.4 Реализация алгоритмов

В листингах 3.1 – 3.3 приведен алгоритм получения гипотез для неоднозначных словоформ и его последовательная и параллельная реализация. В листингах 3.4 – 3.5 приведены реализации дополнительных функций, таких как распределение словоформ файла по массивам и запись гипотез в файл.

Листинг 3.1 – Функция алгоритма получения гипотез для неоднозначных словоформ

```
1 def analyze_words(hypotheses, words):
2     morph = pymorphy2.MorphAnalyzer()
3     for word in words:
4         parsed = morph.parse(word)
5         hypotheses.append([word, [p.tag for p in parsed]])
```

Листинг 3.2 – Функция для запуска алгоритма без вспомогательных потоков

```
1 def no_threads_mode(text, file_name):
2     words = text.split()
3     output = []
4     analyze_words(output, words)
5     write_to_file(output, file_name)
```

Листинг 3.3 – Функция для запуска алгоритма с вспомогательными потоками

```
def with_threads_mode(text, file_name, n):
    words_arr = split_words(text.split(), n)
    res = [[] for _ in range(n)]
    threads = []
    for i, words in enumerate(words_arr):
        thread = threading.Thread(target=analyze_words,
            args=(res[i], words))
        threads.append(thread)

    for thread in threads:
        thread.start()

    for thread in threads:
        thread.join()

    output = sum(res, [])
    write_to_file(output, file_name)
```

Листинг 3.4 – Функция для разбиения слов файла на массивы

```
def split_words(text, n):
    word_lists = [[] for _ in range(n)]
    for i, word in enumerate(text):
        index = i % n
        word_lists[index].append(word)
    return word_lists
```

Листинг 3.5 – Функция для записи результата в файл

```
def write_to_file(data, file_name):
    with open(file_name, 'w', encoding='utf-8') as file:
        for word in data:
            file.write(f'{word}\n')
```

3.5 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для вышеизложенных алгоритмов. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Словоформа	**Результат последовательного**	**Результат параллельного**
'атлас'	('NOUN,inan,masc sing,nomn')	('NOUN,inan,masc sing,nomn')
'запись'	('NOUN,inan,femn sing,nomn')	('NOUN,inan,femn sing,nomn')
'тир'	('NOUN,inan,masc sing,accs')	('NOUN,inan,masc sing,accs')

Вывод

В данном разделе были рассмотрены средства реализации, был представлен листинг реализаций алгоритма получения гипотез для неоднозначных словоформ. Также было проведено тестирование реализаций алгоритма.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программ, проведение исследования и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени:

— Процессор: 11th Gen Intel(R) Core(TM) i5-1135G7 2.42 ГГц.

— Количество ядер: 4 физических и 8 логических ядер.

— Оперативная память: 16 ГБайт.

— Операционная система: Windows 11 Домашняя 64-разрядная система версии 22H2 [7].

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.2 Демонстрация работы программы

Программа получает на вход номер режима и имя файла, в случае режима тестирования имя файла не требуется, а в случае многопоточного режима требуется еще ввести количество потоков. На выходе составляется файл с гипотезами или файл с замерами в зависимости от режима.

На рисунках 4.1 – 4.2 представлена демонстрация работы программы.

```
1. Режим без вспомогательных потоков
2. Многопоточный режим
3. Режим замеров
Выберите режим: 2
Введите название файла с расширением: data/input100.txt
Введите количество потоков: 2
```

Рисунок 4.1 – Демонстрация интерфейса программы

```
['атлас', [OpencorporaTag('NOUN,inan,masc sing,nomn'), OpencorporaTag('NOUN,inan,masc sing,accs')]]
['вера', [OpencorporaTag('NOUN,inan,femn sing,nomn'), OpencorporaTag('NOUN,anim,femn,Name sing,nomn')]]
['запись', [OpencorporaTag('NOUN,inan,femn sing,nomn'), OpencorporaTag('NOUN,inan,femn sing,accs')]]
['пир', [OpencorporaTag('NOUN,inan,masc sing,nomn'), OpencorporaTag('NOUN,inan,masc sing,accs')]]
['край', [OpencorporaTag('NOUN,inan,masc sing,nomn'), OpencorporaTag('NOUN,inan,masc sing,accs')]]
['вес', [OpencorporaTag('NOUN,inan,masc sing,accs'), OpencorporaTag('NOUN,inan,masc sing,nomn')]]
['медь', [OpencorporaTag('NOUN,inan,femn sing,nomn'), OpencorporaTag('NOUN,inan,femn sing,accs')]]
['пенс', [OpencorporaTag('NOUN,inan,masc sing,nomn'), OpencorporaTag('NOUN,inan,masc sing,accs')]]
['сугроб', [OpencorporaTag('NOUN,inan,masc sing,nomn'), OpencorporaTag('NOUN,inan,masc sing,accs')]]
```

Рисунок 4.2 – Демонстрация результата работы программы

4.3 Временные характеристики

Результаты эксперимента замеров по времени приведены в таблице 4.1. Замеры проводились по 10 раз для набора значений потоков 0, 1, 2, 4, 6, 8, 16, 32, 64, где значение количества потоков 0 соответствует однопоточной программе, а значение 1 — программе, создающей один дополнительный поток, выполняющий все вычисления.

Таблица 4.1 – Результаты нагрузочного тестирования

Размер, Кб	**Время, с**							
	0	**1**	**2**	**4**	**8**	**16**	**32**	**64**
100	1.16	1.22	0.95	1.06	1.29	1.73	2.61	4.41
200	2.01	2.10	1.98	1.90	2.14	2.56	3.47	5.28
300	3.01	3.16	2.75	2.75	2.94	3.39	4.30	6.26
400	4.46	4.52	3.78	3.73	3.83	4.33	5.18	6.98
500	5.61	5.71	4.81	4.52	4.69	5.10	6.02	7.82

По таблице 4.1 был построен график 4.3 для последовательного алгоритма и параллельного алгоритма с одним вспомогательным потоком. Из полученных результатов можно сделать вывод, что однопоточный процесс работает быстрее процесса, создающего вспомогательный поток для обработки всех документов. Это связано с дополнительными временными затратами на создание потока и передачи ему необходимых аргументов.

По таблице 4.1 был также построен график 4.4 для варьирующегося числа вспомогательных потоков. Наилучший результат по времени для всех значений количества документов показал процесс с 4 дополнительными потоками, выполняющими вычисления. Рекомендуется использовать на данной архитектуре ЭВМ число дополнительных потоков равное числу физических ядер устройства. Когда количество потоков становится больше, чем количество физических ядер процессора, затраты на управление этими потоками начинают превышать преимущества от использования многопоточности.

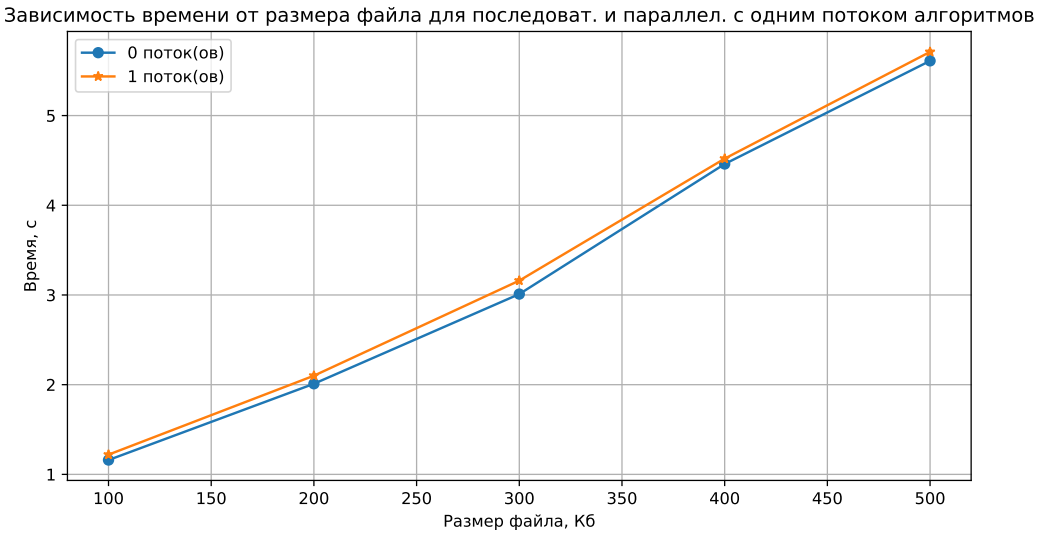

Рисунок 4.3 – График зависимости времени от размера файла для последовательного алгоритма и параллельного алгоритма с одним вспомогательным потоком

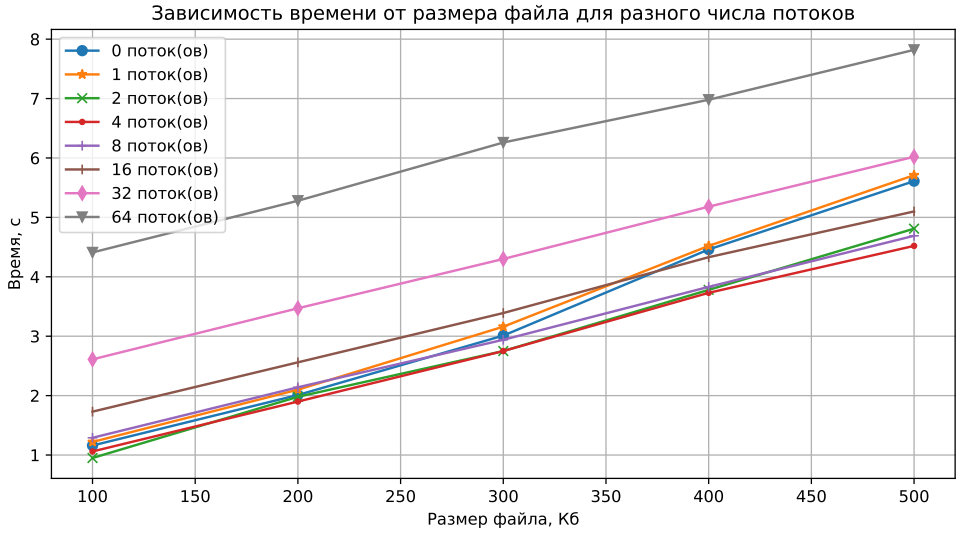

Рисунок 4.4 – График зависимости времени от размера файла для алгоритмов с варьирующимся числом вспомогательных потоков

4.4 Вывод

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

В результате экспериментов было выявлено, что использование многопоточности может уменьшить время выполнения реализации алгоритма по сравнению с однопоточной программой при условии использования подходящего количества потоков. Таким образом, использование дополнительных потоков может как ускорить выполнение процесса по сравнению с однопоточным процессом (для 4 потоков), так и увеличить время выполнения (для 64 потоков). Рекомендуется использовать на данной архитектуре ЭВМ число дополнительных потоков, равное числу физических ядер устройства.

Заключение

В ходе выполнения лабораторной работы было выявлено, что в результате использования многопоточной реализации время выполнения процесса может как улучшиться, так и ухудшиться в зависимости от количества используемых потоков.

Когда количество потоков становится больше, чем количество физических ядер процессора, затраты на управление этими потоками начинают превышать преимущества от использования многопоточности. Это приводит к увеличению времени выполнения программы, особенно по сравнению с лучшим результатом времени выполнения, достигнутым при использовании 4 потоков.

Цель, которая была поставлена в начале лабораторной работы была достигнута, а также в ходе выполнения лабораторной работы были решены следующие задачи.

1) Описаны понятие параллельных вычислений и работа с библиотекой *pymorphy2*.

2) Реализованы последовательный и параллельный алгоритм работы.

3) Замерено время реализации.

4) Проведен анализ затрат работы программы по времени.

Список использованных источников

1 Stoltzfus Justin. Multithreading. — Режим доступа: https://www.techopedia.com/definition/24297/multithreading-computer-architecture (дата обращения: 12.11.2023).

2 У. Ричард Стивенс Стивен А. Раго. UNIX. Профессиональное программирование. 3-е издание. — СПб.: Питер, 2018. с. 994.

3 Документация по pymorphy2 [Электронный ресурс]. — Режим доступа: https://pymorphy2.readthedocs.io/en/stable/index.html (дата обращения: 12.11.2023).

4 Документация по python [Электронный ресурс]. — Режим доступа: https://docs.python.org/3/ (дата обращения: 13.11.2023).

5 Документация по time [Электронный ресурс]. — Режим доступа: https://docs.python.org/3/library/time.html (дата обращения: 13.11.2023).

6 Документация по threading [Электронный ресурс]. — Режим доступа: https://docs.python.org/3/library/threading.html (дата обращения: 13.11.2023).

7 Windows 11 [Электронный ресурс]. — Режим доступа: https://www.microsoft.com/ru-ru/software-download/windows11 (дата обращения: 12.11.2023).