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 Виноградов А. О.

Группа ИУ7-56Б

Оценка (баллы)

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

Москва — 2022 г.

Содержание

Введение

По мере развития вычислительных систем программисты столкнулись с необходимостью производить параллельную обработку данных для улучшения отзывчивости системы, ускорения производимых вычислений и рационального использования вычислитлельных мощностей. Благодаря развитию процессоров стало возможным использовать один процессор для выполнения нескольких параллельных операций, что дало начало термину «многопоточность».

Целью данной лабораторной работы является изучение принципов и получение навыков организации параллельного выполнения операций на примере сервера раздачи статической информации.

Задачи данной лабораторной работы:

1) изучение основы распараллеливания вычислений;
2) проведение сравнительного анализа по времени работы алгоритма выделения наиболее информативных терминов набора документов на основе документной частоты с использованием многопоточности и без нее;
3) проведение анализа зависимости времени выполнения выше озвученного алгоритма от количества потоков, участвующих в обработке;
4) описание и обоснование полученных результатов в отчете о выполненной лабораторной работе.

1 Аналитическая часть

В данном разделе была представлена информация о многопоточности и исследуемом алгоритме выделения терминов для классификации.

1.1 Многопоточность

Многопоточность [1] — это способность центрального процессора (ЦП) обеспечивать одновременное выполнение нескольких потоков в рамках использования ресурсов одного процессора. Поток – последовательность инструкций, которые могут исполняться параллельно с другими потоками одного и того же породившего их процесса.

Процессом называют программу в ходе своего выполнения. Каждый процесс может состоять из одного или нескольких потоков. Процесс завершается тогда, когда все его потоки заканчивают работу. Каждый поток в операционной системе является задачей, которую должен выполнить процессор. Сейчас большинство процессоров умеют выполнять несколько задач на одном ядре, создавая дополнительные, виртуальные ядра, или же имеют несколько физических ядер. Такие процессоры называются многоядерными.

При использовании потоков возникает проблема множественного доступа к информации. Необходимо обеспечить невозможность записи в одну ячейку памяти из нескольких потоков одновременною

Таким образом, необходимо использовать примитивы синхронизации обращения к данным. Одним из таких примитивов является мьютекс. Мьютекс может быть захвачен для работы в монопольном режиме или освобождён. Так, если 2 потока одновременно пытаются захватить мьютекс, захват произведет только один, а другой будет ждать освобождения..

Критическая секция — набор инструкций, выполняемый между захватом и высвобождением мьютекса. Поскольку в то время, пока мьютекс захвачен, остальные потоки, требующие выполнения критической секции, ждут осво-

бождения мьютекса, требуется разрабатывать программное обеспечение таким образом, чтобы критическая секция была минимизирована по времени выполнения для сокращения задержек.

1.2 Документная частота

Самая простая и вполне эффективная [2] техника оценки «важности терминов для классификации» основана на наблюдении того, что значительное число терминов коллекции встречаются в малом числе документов, а наибольшую информативность имеют термины со средней или даже высокой частотой, если предварительно были удалены стоп-слова. Данная техника может применяться как единственная, так и предшествовать другой технике отбора признаков.

Таким образом, алгоритм выделения наиболее информативных терминов (из выборки документов) состоит на подсчете документной частоты (DF) — количества документов, в которых встречается термин t_k и выбором терминов с наибольшими значениями DF.

В данной лабораторной работе проводится распараллеливание алгоритма выделения терминов из выборки текстов на основе документной частоты. Для этого все документы поровну распределяются между всеми потоками.

В качестве одного из аргументов каждый поток получает выделенный для него массив счетчиков DF длины L, где L — это мощность алфавита. Так как каждый массив передается в монопольное использование каждому потоку, не возникает конфликтов доступа к разделяемым ячейкам памяти, следовательно, в использовании средства синхронизации в виде мьютекса нет нужды.

Вывод

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

2 Конструкторская часть

В данном разделе разработаны схемы реализаций алгоритма выделения наиболее информативных терминов из выборки документов.

2.1 Разработка алгоритма

На рисунках 2.1 – 2.3 приведены схемы однопоточной и многопоточной реализаций рассматриваемого алгоритма.

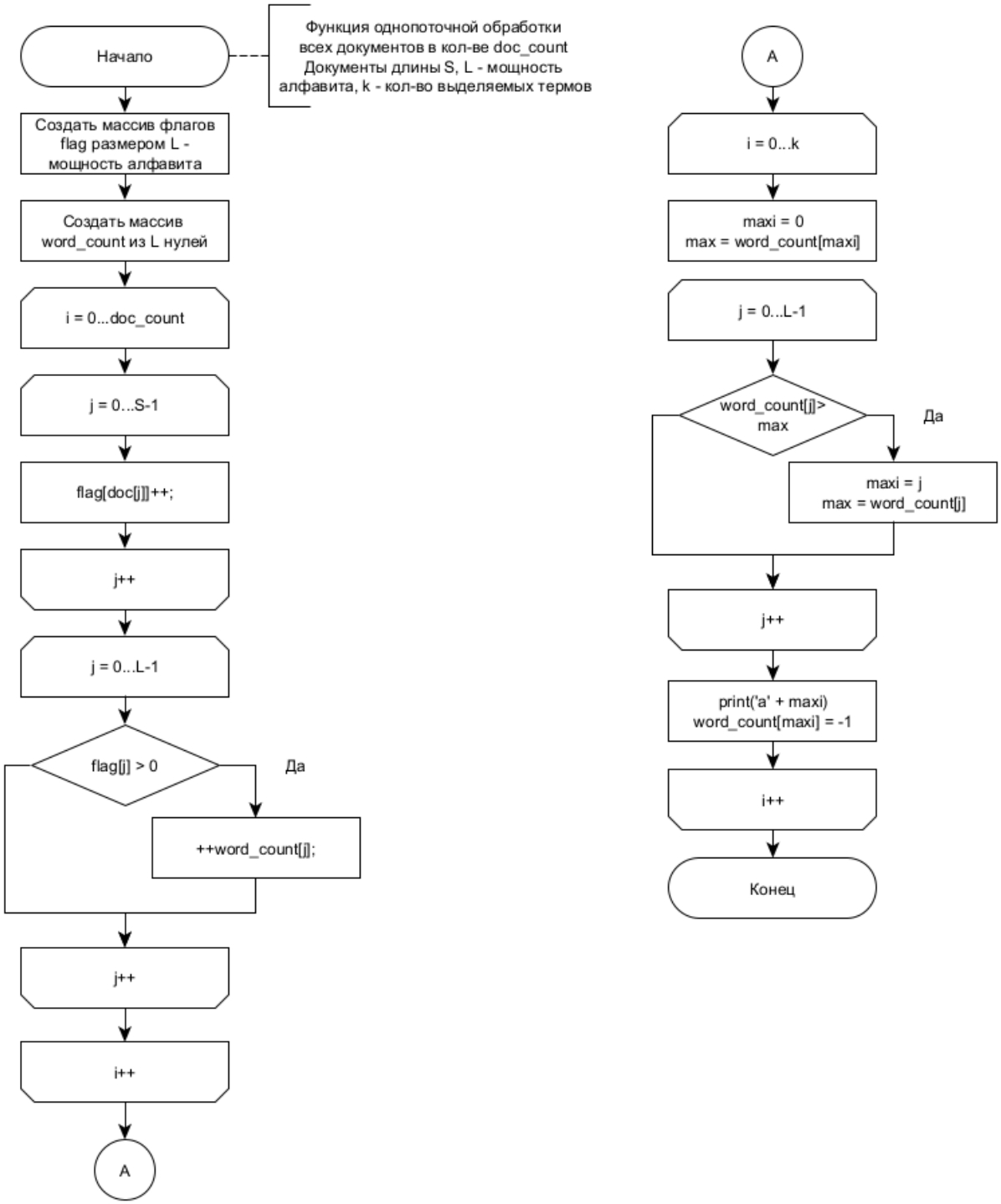

Рисунок 2.1 – Схема однопоточного алгоритма выделения терминов из выборки документов на основе документной частоты

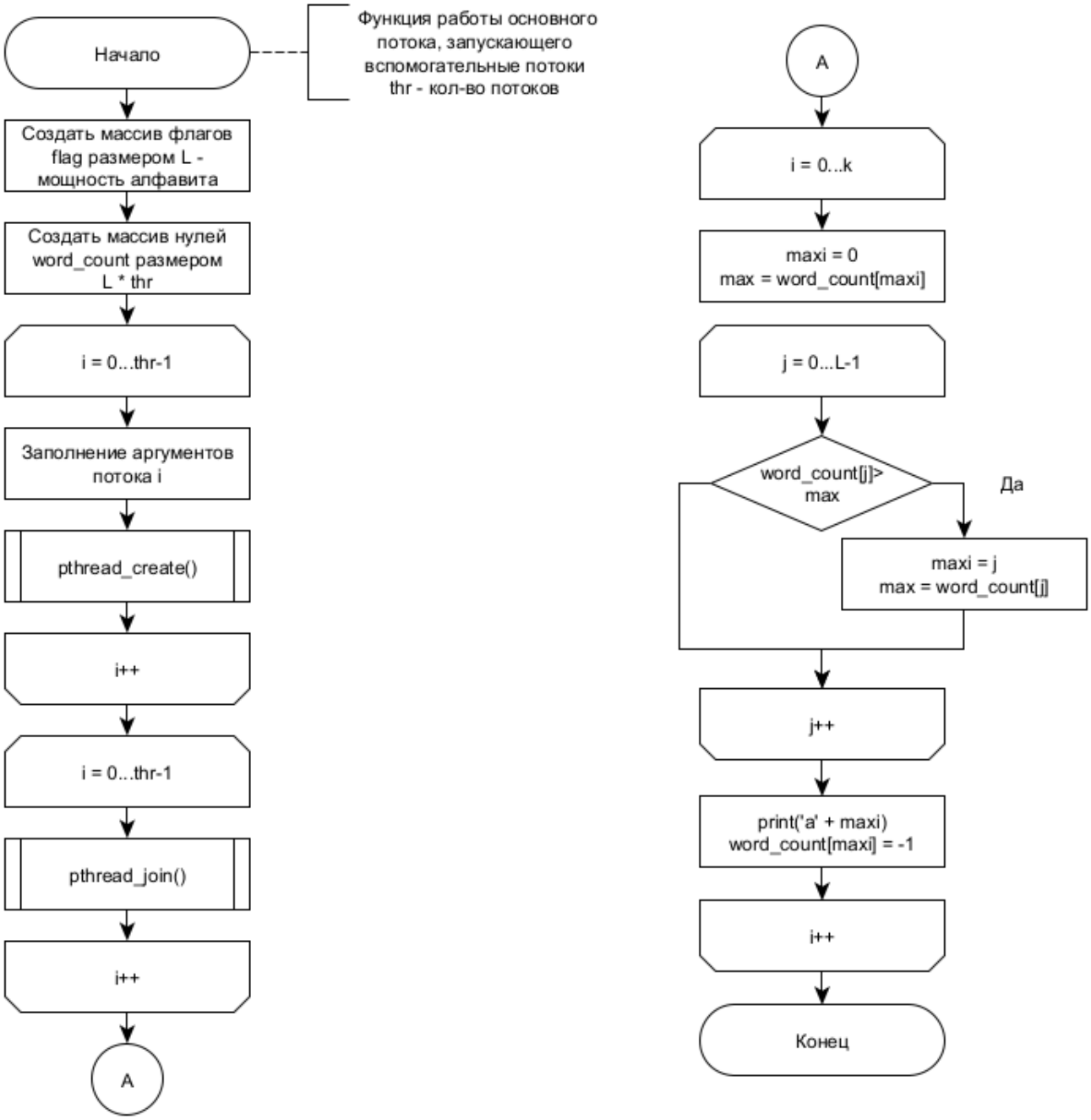

Рисунок 2.2 – Схема алгоритма работы основного потока, запускающего вспомогательные потоки

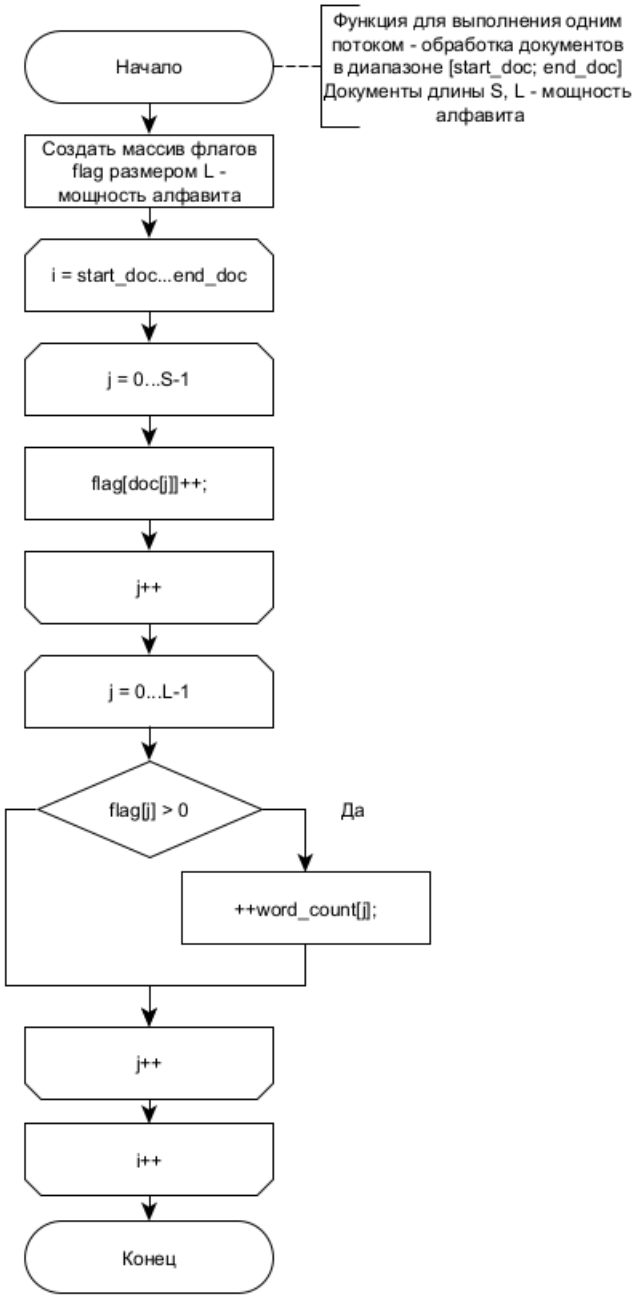

Рисунок 2.3 – Схема алгоритма работы одного вспомогательного потока

Вывод

В данном разделе разработаны схемы реализаций рассматриваемого алгоритма.

3 Технологическая часть

В данном разделе рассмотрены средства реализации, а также представлены листинги реализации алгоритма выделения наиболее информативных терминов, по метрике DF, из выборки документов.

3.1 Средства реализации

Для данной работы был выбран язык C [3]. Для данной лабораторной работы требуются инструменты для работы с массивами и потоками, замеров времени работы выполняемой программы. Все перечисленные инструменты присутствуют в выбранном языке программирования.

Для работы с потоками использовались функции заголовочного файла <pthread.h> [4]. Для работы с сущностью вспомогательного потока необходимо воспользоваться функцией $pthread\_create()$ для создания потока и указания функции, которую начнет выполнять созданный поток. Далее при помощи вызова $pthread\_join()$ необходимо (в рамках данной лабораторной работы) дождаться завершения всех вспомогательных потоков, чтобы в главном потоке обработать результаты их работы.

3.2 Реализация алгоритмов

В листингах 3.1 – 3.4 приведены реализации алгоритма выделения наиболее информативных терминов из выборки документов. В качестве терминов в данной реализации рассматриваются односложные слова из букв латинского алфавита. В качестве документов рассматриваются строки, состоящие из таких слов.

Листинг 3.1 – Однопоточная функция выделения наиболее употребляемых термов

```
1  void one_thread(int docs_cnt, int doc_size)
2  {
3      char *docs = malloc(sizeof(char) * docs_cnt * doc_size);
4      set_frequency(docs, docs_cnt, doc_size);
5
6      int word_count[LETTERS_CNT];
7      for (int i = 0; i < LETTERS_CNT; ++i)
8      word_count[i] = 0;
9
10     for (int i = 0; i < docs_cnt; ++i)
11     {
12         for (int j = 0; j < doc_size; ++j)
13         printf("%c", docs[i * doc_size + j]);
14         printf("\n");
15     }
16
17     for (int i = 0; i < docs_cnt; ++i)
18     {
19         int *flag_arr = allocate_flag_arr(LETTERS_CNT);
20         for (int j = 0; j < DOCSIZE; ++j)
21         ++(flag_arr[(docs)[i*DOCSIZE + j] - 'a']);
22
23         for (int i = 0; i < LETTERS_CNT; ++i)
24         {
25             if (flag_arr[i])
26             ++word_count[i];
27         }
28         free_flag_arr(flag_arr);
29     }
30     printf("word_count:\n");
31     for (int i = 0; i < LETTERS_CNT; ++i)
32         printf("%d␣", word_count[i]);
33     printf("\nMost␣frequent␣terms:");
34     for (int j = 0; j < 5; j++)
35     {
36         int maxi = 0;
37         int max = word_count[maxi];
38         for (int i = 0; i < LETTERS_CNT; ++i)
39         {
```

Листинг 3.2 – Однопоточная функция выделения наиболее употребляемых термов

```
            if (word_count[i] > max)
            {
                max = word_count[i];
                maxi = i;
            }
        }
        printf("␣%c", 'a' + maxi);
        word_count[maxi] = -1;
    }
    free(docs);
```

Листинг 3.3 – Функция работы основного потока запускающего вспомогательные потоки

```
void with_threads(int threads, char *docs, int docs_cnt)
    int *word_count = calloc(threads * LETTERS_CNT, sizeof(int));
    pthread_t *tid = malloc(threads * sizeof(pthread_t));
    thread_args_t *args = malloc(threads * sizeof(thread_args_t));
    if (tid && args)
    {
        int delta_doc = docs_cnt / threads;
        int last = docs_cnt % threads;
        int start = 0;
        int end = delta_doc + last;
        for (int thread = 0; thread < threads; ++thread)
        {
            (args + thread)->start_doc = start;
            (args + thread)->end_doc = end;
            // (args + thread)->mutex = mutex;
            (args + thread)->docs = docs;
            (args + thread)->word_count = word_count + thread *
               LETTERS_CNT;
            pthread_create(tid + thread, NULL, &thread_work, args +
               thread);
            start = end;
            end += delta_doc;
        }
        for (int thread = 0; thread < threads; ++thread)
        pthread_join(tid[thread], NULL);
    }
}
```

Листинг 3.4 – Функция работы одного вспомогательного потока

```
void *thread_work(void *args)
{
    thread_args_t *arg = args;

    int start = arg->start_doc;
    int end = arg->end_doc;
    char *docs = arg->docs;
    int *word_count = arg->word_count;
    for (int i = start; i < end; ++i)
    {
        int *flag_arr = allocate_flag_arr(LETTERS_CNT);
        for (int j = 0; j < DOCSIZE; ++j)
        ++(flag_arr[(docs)[i*DOCSIZE + j] - 'a']);

        for (int i = 0; i < LETTERS_CNT; ++i)
        {
            if (flag_arr[i])
            ++(word_count[i]);
        }
        free_flag_arr(flag_arr);
    }

    return NULL;
}
```

Вывод

В данном разделе были рассмотрены средства реализации, а также представлен листинг реализации алгоритма выделения наиболее информативных терминов из выборки документов.

4 Исследовательская часть

В данном разделе приведены технические характеристики устройства, на котором проводилось измерение времени работы ПО, а также результаты замеров времени.

4.1 Технические характеристики

Ниже приведены технические характеристики устройства, на котором было проведено измерение времени работы ПО:

- операционная система Windows 10 Домашняя Версия 21H1 [5] x86_64;
- оперативная память 8 Гбайт 2133 МГц;
- процессор Intel Core i5-8300H с тактовой частотой 2.30 ГГц [6], 4 физических ядра, 8 логических ядер.

4.2 Время выполнения реализации алгоритма

Для замеров времени использовалась функция получения значения системных часов *clock_gettime*() [7]. Функция применялась два раза — в начале и в конце измерения времени, значения полученных временных меток вычитались друг из друга для получения времени выполнения программы.

Документы заполнялись случайными буквами латинского алфавита. Замеры проводились по 1000 раз для набора значений количества потоков 0, 1, 2, 4, 8, 16, 32, 64, где значение количества потоков *0* соответствует однопоточной программе, а значение *1* — программе, создающей один дополнительный поток, выполняющий все вычисления. В таблице 4.1 представлены замеры времени выполнения программы в зависимости от количества потоков.

На рисунках 4.1 – 4.2 приведена графическая интерпретация результатов замеров.

Таблица 4.1 – Результаты нагрузочного тестирования (в мкс)

Кол-во документов	Кол-во потоков							
	0	1	2	4	8	16	32	64
2048	515	761	430	416	525	887	1674	3328
2560	648	924	521	462	543	910	1678	3339
3072	778	1066	621	511	590	950	1730	3398
3584	927	1204	700	552	633	984	1768	3456
4096	1042	1346	809	607	656	1016	1807	3487
4608	1181	1484	919	643	706	1055	1853	3493
5120	1323	1606	1016	684	732	1081	1862	3514
5632	1426	1738	1081	734	765	1104	1893	3544
6144	1571	1886	1164	825	816	1129	1919	3598
6656	1688	2010	1247	861	851	1159	1953	3606
7168	1824	2135	1341	968	879	1249	1972	3654
7680	1954	2258	1412	963	937	1215	1996	3666
8192	2066	2450	1504	1053	956	1260	2014	3711

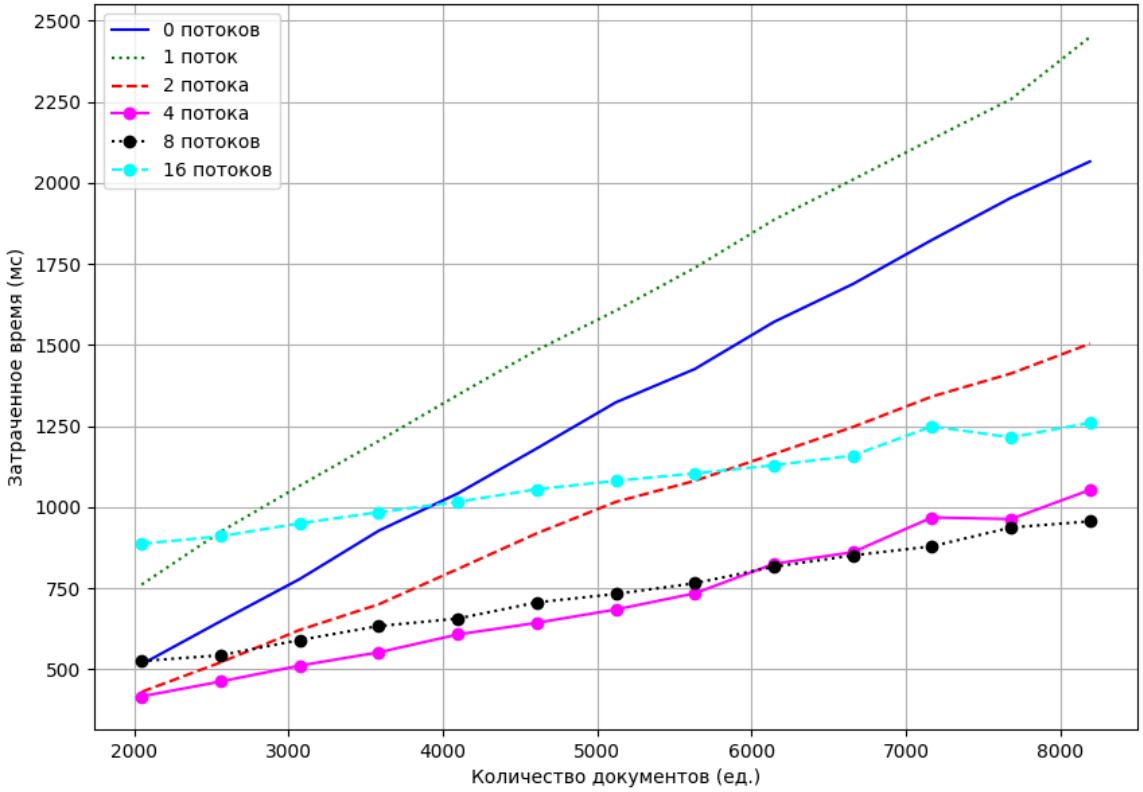

Рисунок 4.1 – Результаты замеров времени работы реализаций алгоритма с разным количеством потоков в зависимости от количества документов

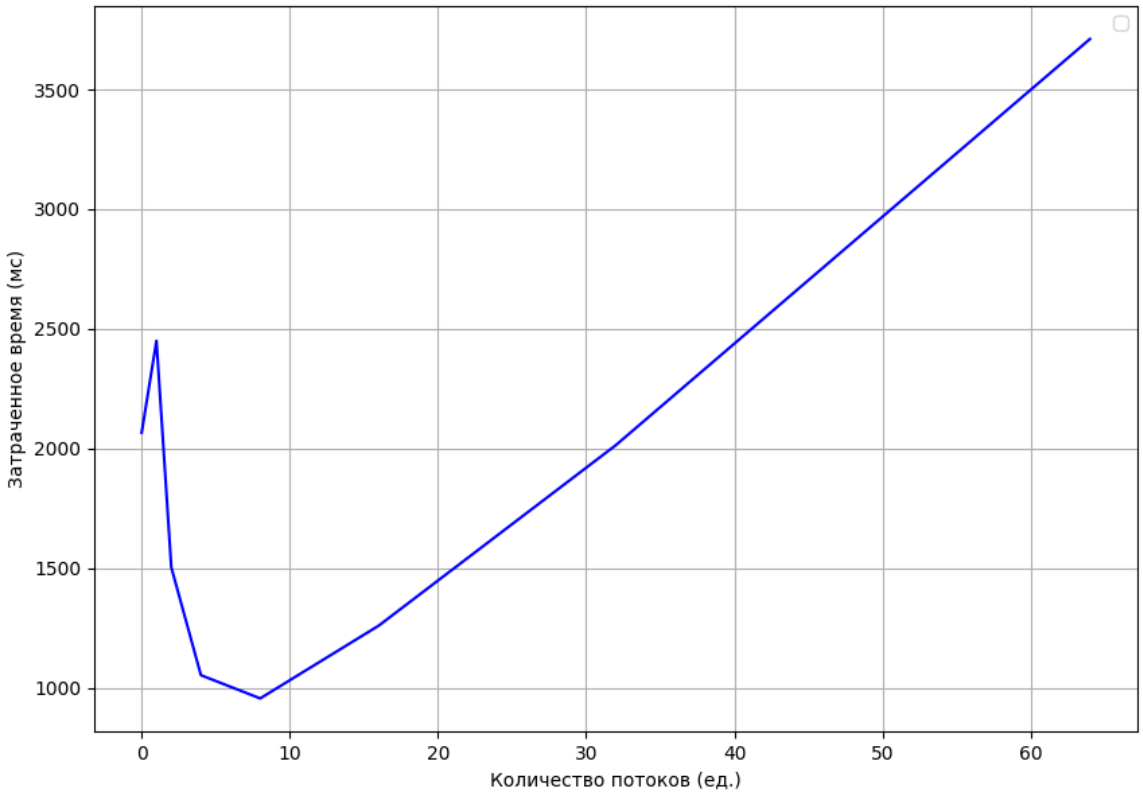

Рисунок 4.2 – Результаты замеров времени работы реализации алгоритма для 8192 документов в зависимости от количества потоков

Из полученных результатов можно сделать вывод, что однопоточный процесс работает быстрее процесса, создающего один отдельный поток для обработки всех документов. Это связано с дополнительными временными затратами на создание потока и передачи ему необходимых аргументов.

Наилучший результат по времени для всех значений количества документов показал процесс с 8 дополнительными потоками, выполняющими вычисления. Рекомендуется использовать на данной архитектуре ЭВМ число дополнительных потоков равное числу логических ядер устройства.

Для числа потоков, большего 8, затраты на содержание потоков превышают преимущество от использования многопоточности, и функция времени от количества потоков начинает расти.

Вывод

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.

Выборка из результатов замеров времени (для 8192 документов):

- однопоточный процесс — 2066 мкс;
- один дополнительный поток, выполняющий все вычисления — 2450 мкс;
- 8 потоков (лучший результат) — 956 мкс, что в 2,16 раз быстрее выполнения однопоточного процесса;
- 64 потока (худший результат) — 3711 мкс, что в 1,80 раз быстрее выполнения однопоточного процесс.

Таким образом, использование дополнительных потоков может как ускорить выполнение процесса по сравнению с однопоточным процессом (в 2,16 раз для 8 потоков), так и увеличить время выполнения (в 1,80 раз для 64 потоков).

Рекомендуется использовать на данной архитектуре ЭВМ число дополнительных потоков равное числу логических 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.

Выборка из результатов замеров времени (для 8192 документов):

- однопоточный процесс — 2066 мкс;
- один дополнительный поток, выполняющий все вычисления — 2450 мкс;
- 8 потоков (лучший результат) — 956 мкс, что в 2,16 раз быстрее выполнения однопоточного процесса;
- 16 потоков — 1260 мкс, что в 1,64 раз быстрее выполнения однопоточного процесса;
- 32 потока — 2014 мкс, что сравнимо с временем выполнения однопоточного процесса;
- 64 потока (худший результат) — 3711 мкс, что в 1,80 раз медленнее выполнения однопоточного процесс.

При слишком большом значении потоков (более 8 для устройства, на котором проводилось тестирование) затраты на содержание потоков превышают преимущество от использования многопоточности и время выполнения по сравнению с лучшим результатом (для 8 потоков) растут.

Пусть x — количество потоков, MAX — максимальное количество потоков, разрешенное для одного процесса. Влияние количества потоков на время выполнения по сравнению с однопоточным процессом:

- $x \in [2; 32)$ — улучшение времени выполнения;
- $x \in 32$ — влияние на время выполнения незначительно;

– $x \in (32, MAX)$ — ухудшение времени выполнения.

Цель, поставленная в начале работы, была достигнута. Кроме того были достигнуты все поставленные задачи:

– были изучены основы распараллеливания вычислений;

– был проведен сравнительный анализ по времени работы алгоритма выделения наиболее информативных терминов набора документов на основе документной частоты с использованием многопоточности и без нее;

– был проведен сравнительный анализ зависимости времени выполнения выше озвученного алгоритма от количества потоков, участвующих в обработке;

– был подготовлен отчет о лабораторной работе.

Список использованных источников

[1] Stoltzfus J. Multithreading. – Techopedia - Janalta Interactive Inc. [Электронный ресурс], 2022. URL: Режим доступа: https://www.techopedia.com/definition/24297/multithreading-computer-architecture (дата обращения: 11.10.2021).

[2] Большакова Е.И., Клышинский Э.С., Ландэ Д.В., Носков А.А., Пескова О.В., Ягунова Е.В. Автоматическая обработка текстов на естественном языке и компьютерная лингвистика. — М.: МИЭМ, 2011. — С. 171–173.

[3] C language [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/c/language (дата обращения: 10.11.2022).

[4] pthread.h - threads [Электронный ресурс]. Режим доступа: https://pubs.opengroup.org/onlinepubs/007908799/xsh/pthread.h.html (дата обращения: 10.11.2022).

[5] Windows [Электронный ресурс]. Режим доступа: https://www.microsoft.com/en-us/windows (дата обращения: 01.10.2022).

[6] Процессор Intel Core i5 [Электронный ресурс]. Режим доступа: https://www.intel.com/processors/core/i5/docs (дата обращения: 01.10.2022).

[7] Определение текущего времени с высокой точностью [Электронный ресурс]. Режим доступа: http://all-ht.ru/inf/prog/c/func/clock_gettime.html (дата обращения: 10.11.2022).