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 № 5 по курсу «Анализ алгоритмов»

Тема Конвейерная обработка данных

Студент Виноградов А. О.

Группа ИУ7-56Б

Оценка (баллы)

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

Москва — 2023 г.

Содержание

Введение

Конвейерная обработка позволяет ускорить обработку данных благодаря использованию принципов параллельности. Идея взята из работы реальных конвейерных лент — данные поступают на обработку и проходят несколько независимых друг от друга стадий, что позволяет одновременно обрабатывать несколько единиц данных.

Целью данной работы является применение принципов конвейерной обработки данных.

Задачи данной лабораторной работы:

1) описание основ конвейерной обработки данных;
2) реализация алгоритма выделения наиболее информативных терминов набора документов на основе документной частоты;
3) разработка программного продукта, позволяющего выполнять как последовательную, так и конвейерную реализацию озвученного алгоритма;
4) проведение сравнительного анализа временной эффективности линейной и параллельной конвейерной реализаций алгоритма;
5) описание и обоснование полученных результатов в отчете о выполненной лабораторной работе.

1 Аналитическая часть

В данном разделе рассмотрена конвейерная обработка данных, представлена информация об исследуемом алгоритме выделения терминов для классификации.

1.1 Конвейерная обработка данных

Конвейер [1] (англ. conway) — организация вычислений, при которой увеличивается количество выполняемых инструкций за единицу времени за счет использования принципов параллельности. Конвейерная обработка в общем случае основана на разделении подлежащей исполнению функции на более мелкие части, называемые ступенями, и выделении для каждой из них отдельного блока аппаратуры. Так, обработку любой машинной команды можно разделить на несколько этапов, организовав передачу данных от одного этапа к следующему. При этом конвейерную обработку можно использовать для совмещения этапов выполнения разных команд. Производительность при этом возрастает, благодаря тому, что одновременно на различных ступенях конвейера выполняется несколько команд. Конвейерная обработка такого рода широко применяется во всех современных быстродействующих процессорах.

Конвейеризация позволяет увеличить пропускную способность процессора (количество команд, завершающихся в единицу времени), но она не сокращает время выполнения отдельной команды. В действительности она даже несколько увеличивает время выполнения каждой команды из-за накладных расходов, связанных с хранением промежуточных результатов. Однако увеличение пропускной способности означает, что программа будет выполняться быстрее по сравнению с простой, неконвейерной схемой.

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 качестве операций, выполняющихся на конвейере, взяты следующие:

1) создание заданного количества текстовых документов для последующей обработки;

2) обсчет документной частоты DF;

3) вывод наиболее часто употребляемых терминов в новый файл.

Ленты конвейера будут передавать друг другу заявки. Первый этап будет формировать заявку, которая будет передаваться от этапа к этапу.

Заявка должна содержать:

– матрицу строк, представляющую собой набор документов;

– словарь, содержащий документную частоту для каждого обнаруженного слова;

– временные отметки.

Вывод

В данном разделе была теоретически разобрана конвейерная обработка данных, представлена информация об исследуемом алгоритме выделения терминов для классификации.

2 Конструкторская часть

В данном разделе разработаны последовательная и параллельная реализации работы стадий конвейера.

2.1 Разработка алгоритмов

На рисунке 2.1 представлен последовательный алгоритм работы стадий конвейера.

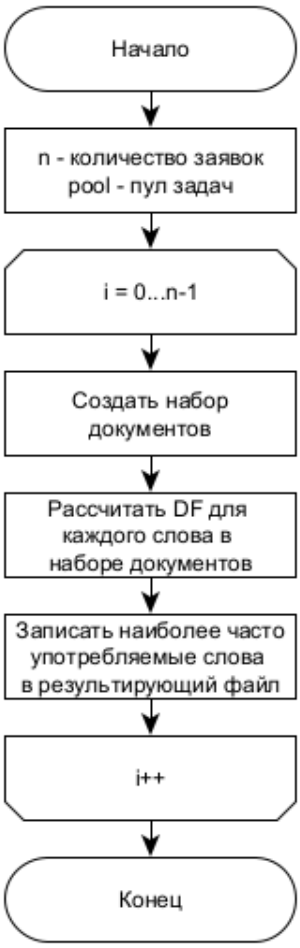

Рисунок 2.1 – Схема последовательной работы алгоритма

Параллельная работа будет реализована с помощью потоков, где каждый поток отвечает за свою стадию обработки. Данные от потока к потоку будут передаваться через очереди. Так как потоков, как и стадий обработки, три, то необходимо две очереди между потоками.

Для получения доступа к изменению содержимого очереди необходимо использовать примитив синхронизации — мьютекс. В противном случае, данные в очереди могут исказится. Так как очереди две, то и мьютексов будет два.

На рисунке 2.2 представлена схема главного потока при параллельной работе стадий конвейера.

Рисунок 2.2 – Схема параллельной работы

На рисунках 2.3 – 2.5 представлены схемы алгоритмов каждого из обработчиков (потоков) при параллельной работе.

Рисунок 2.3 – Схема алгоритма потока 1

Рисунок 2.4 – Схема алгоритма потока 2

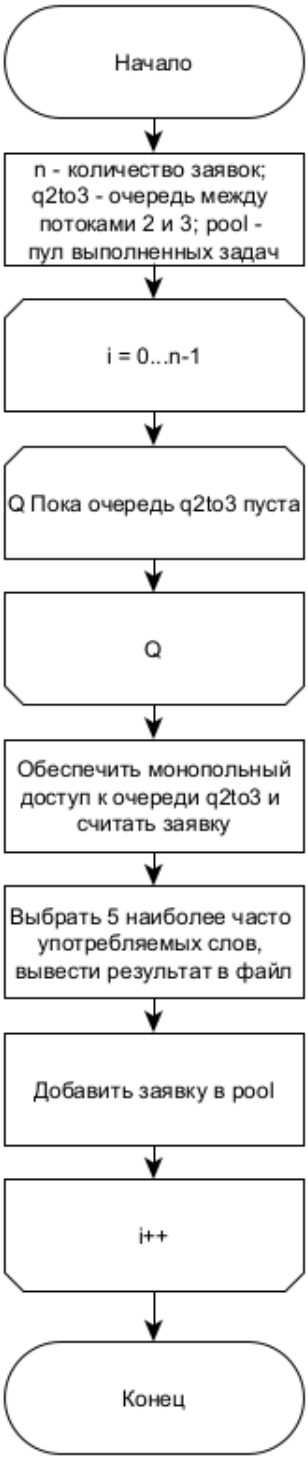

Рисунок 2.5 – Схема алгоритма потока 3

Вывод

В данном разделе разработаны схемы последовательной и параллельной работы стадий конвейера.

3 Технологическая часть

В данном разделе рассмотрены средства реализации, а также представлены листинги последовательной и параллельной работы стадий конвейера.

3.1 Реализация алгоритмов

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

Работа с потоками предоставляется классом *thread* [4].

Работа с мьютексами предоставляется классом *mutex* [5].

Работа с очередями предоставляется классом *queue* [6].

В листингах 3.1–3.2 представлена реализация последовательной работы стадий конвейера.

Листинг 3.1 – Последовательная работа стадий конвейера

```
1  void linear()
2  {
3      int n = TIMES;
4      vector<request *> pool(n);
5      timespec time;
6      for (int i = 0; i < n; ++i)
7      {
8          clock_gettime(CLOCK_REALTIME, &time);
9          request *r = proc_1(DOC_CNT, DOCSIZE);
10         r->p1_start = time;
11         clock_gettime(CLOCK_REALTIME, &(r->p1_end));
```

Листинг 3.2 – Последовательная работа стадий конвейера (окончание)

```
        clock_gettime(CLOCK_REALTIME, &(r->p2_start));
        proc_2(r->files, r->dict);
        clock_gettime(CLOCK_REALTIME, &(r->p2_end));
        clock_gettime(CLOCK_REALTIME, &(r->p3_start));
        proc_3(r->dict, "file.txt");
        clock_gettime(CLOCK_REALTIME, &(r->p3_end));
        pool[i] = r;
    }
    print_pool(pool, "linear.txt");
    for (size_t i = 0; i < pool.size(); ++i)
        delete pool[i];
}
```

В листинге 3.3 представлена реализация параллельной работы стадий конвейера.

Листинг 3.3 – Параллельная работа стадий конвейера

```
void parallel()
{
    int n = TIMES;
    string s = "file.txt";
    vector<request *> pool(n);
    queue<request *> q1to2;
    queue<request *> q2to3;
    thread t_1(thr_1, n, ref(q1to2));
    thread t_2(thr_2, n, ref(q1to2), ref(q2to3));
    thread t_3(thr_3, n, ref(q2to3), ref(pool));
    t_1.join();
    t_2.join();
    t_3.join();
    print_pool(pool, "parallel.txt");
    for (size_t i = 0; i < pool.size(); ++i)
        delete pool[i];
}
```

В листингах 3.4, 3.5, 3.6 представлены реализации каждого потока при параллельной работе.

Листинг 3.4 – Поток для стадии 1

```
void thr_1(int n, queue<request *> &q1to2)
{
    timespec time;
    for (int i = 0; i < n; ++i)
    {
        clock_gettime(CLOCK_REALTIME, &time);
        request *r = proc_1(DOC_CNT, DOCSIZE);
        r->p1_start = time;

        mutex_q1to2.lock();
        clock_gettime(CLOCK_REALTIME, &(r->p1_end));
        q1to2.push(r);
        mutex_q1to2.unlock();
    }
}
```

Листинг 3.5 – Поток для стадии 2

```
void thr_2(int n, queue<request *> &q1to2, queue<request *> &q2to3)
{
    for (int i = 0; i < n; ++i)
    {
        while (q1to2.empty());
        mutex_q1to2.lock();
        request *r = q1to2.front();
        q1to2.pop();
        mutex_q1to2.unlock();
        clock_gettime(CLOCK_REALTIME, &(r->p2_start));
        proc_2(r->files, r->dict);

        mutex_q2to3.lock();
        clock_gettime(CLOCK_REALTIME, &(r->p2_end));
        q2to3.push(r);
        mutex_q2to3.unlock();
    }
}
```

Листинг 3.6 – Поток для стадии 3

```
void thr_3(int n, queue<request *> &q2to3, vector<request *> &pool)
{
    for (int i = 0; i < n; ++i)
    {
        while (q2to3.empty());
        mutex_q2to3.lock();
        request *r = q2to3.front();
        q2to3.pop();
        mutex_q2to3.unlock();
        clock_gettime(CLOCK_REALTIME, &(r->p3_start));
        proc_3(r->dict, "file.txt");
        clock_gettime(CLOCK_REALTIME, &(r->p3_end));
        pool[i] = r;
    }
}
```

В листинге 3.7 представлена реализация первой стадии конвейера — создание набора документов.

Листинг 3.7 – Функция работы одного вспомогательного потока

```
request *proc_1(int doc_count, int word_count)
{
    request *r = new request();
    for (int i = 0; i < doc_count; ++i)
    {
        vector<string> doc;
        for (int j = 0; j < word_count; ++j)
        doc.push_back(gen_random(RAND_LEN));
        r->files.push_back(doc);
    }
    return r;
}
```

В листинге 3.8 представлена реализация второй стадии конвейера — вычисление DF.

Листинг 3.8 – Функция работы одного вспомогательного потока

```
void proc_2(vector<vector<string>> &files, map<string, int> &dict)
{
    for (auto file:files)
    {
        set<string> sett;
        for (auto word : file)
        sett.insert(word);
        for (auto s : sett)
        {
            if (dict.count(s) == 0)
                dict.insert({s, 1});
            else
                ++dict[s];
        }
    }
}
```

В листинге 3.9 представлена реализация третьей стадии конвейера — выборка наиболее употребляемых слов и их запись в выходной файл.

Листинг 3.9 – Функция работы одного вспомогательного потока

```
void proc_3(map<string, int> &dict, string filename)
{
    ofstream myfile; myfile.open (filename);
    for (int j = 0; j < WORDS_TO_CHOOSE; j++)
    {
        string maxi = dict.begin()->first;
        int max = dict.begin()->second;
        for (auto i:dict)
        {
            if (i.second > max)
                max = i.second; maxi = i.first;
        }
        dict[maxi] = -1;
        myfile<<maxi<<' '<<max<<endl;
    }
    myfile.close();
}
```

Вывод

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

4 Исследовательская часть

В данном разделе приведены примеры работы программы, а также проведен сравнительный анализ последовательной и параллельной реализаций при различном количестве заявок.

4.1 Технические характеристики

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

- операционная система Windows 10 Домашняя Версия 21H1 [7] x86_64;
- оперативная память 8 Гбайт 2133 МГц;
- процессор Intel Core i5-8300H с тактовой частотой 2.30 ГГц [8], 4 физических ядра, 8 логических ядер.

4.2 Демонстрация работы программы

На рисунке 4.1 представлен приме работы программы. Пользователь, указывая соответствующие пункты меню, запускает последовательную или параллельную обработку заявок. Результаты работы на экран не выводятся, а записываются в файл $file.txt$. Пример результатов работы программы представлен на рисунке 4.2. Лог программы также на экран не выводится — он записывается в файл. На рисунке 4.3 представлен пример лог–файла.

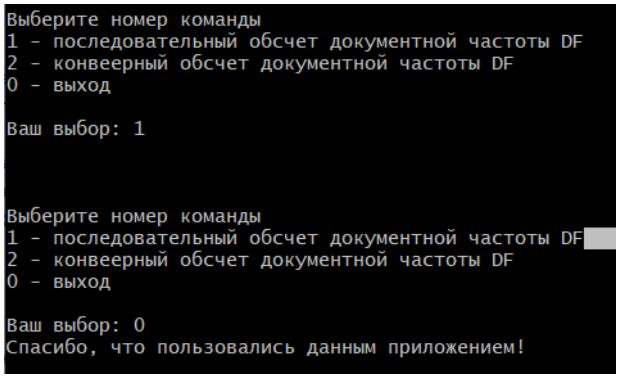

Рисунок 4.1 – Пример работы программы

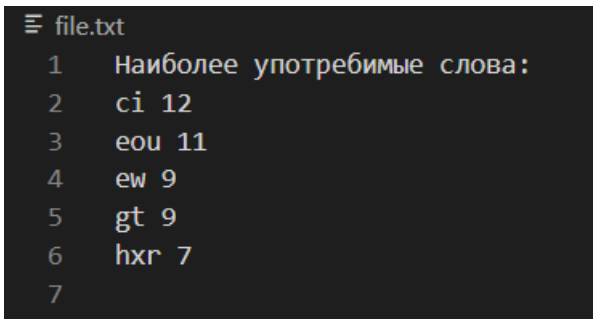

Рисунок 4.2 – Пример результатов работы программы

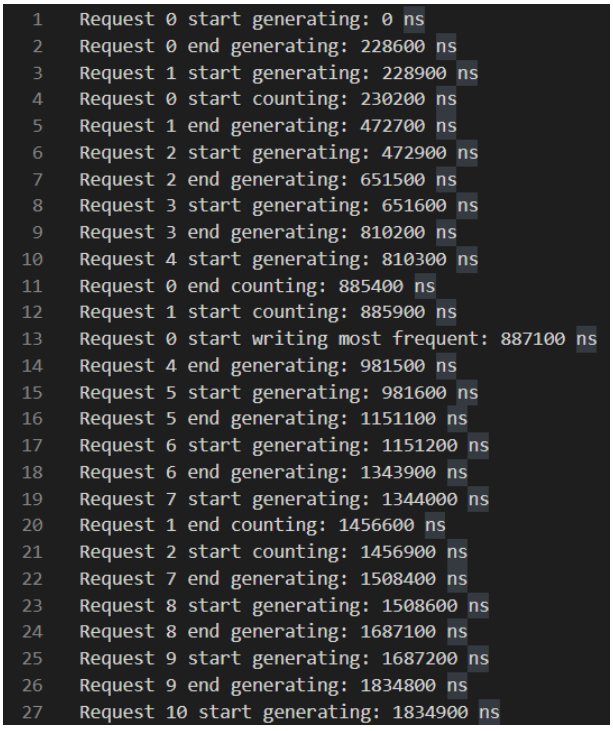

Рисунок 4.3 – Пример лог–файла

4.3 Время выполнения реализации алгоритма

Для замеров времени использовалась функция получения значения системных часов *clock_gettime*() [9]. 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 словами из букв латинского алфавита. Для замеров времени использовались наборы из 30 документов по 20 слов каждый. Замеры проводились по 100 раз для набора значений количества заявок 10, 20, 30, 40, 50, 60, 70, 80, 90, 100. В таблице 4.1 представлены замеры времени выполнения реализаций в зависимости от количества заявок.

Таблица 4.1 – Результаты нагрузочного тестирования (в мс)

Кол-во заявок	Последовательная работа	Параллельная работа
10	32	8
20	65	16
30	98	24
40	131	32
50	166	39
60	198	49
70	233	56
80	267	64
90	301	73
100	332	81

На рисунке 4.4 приведена графическая интерпретация результатов замеров.

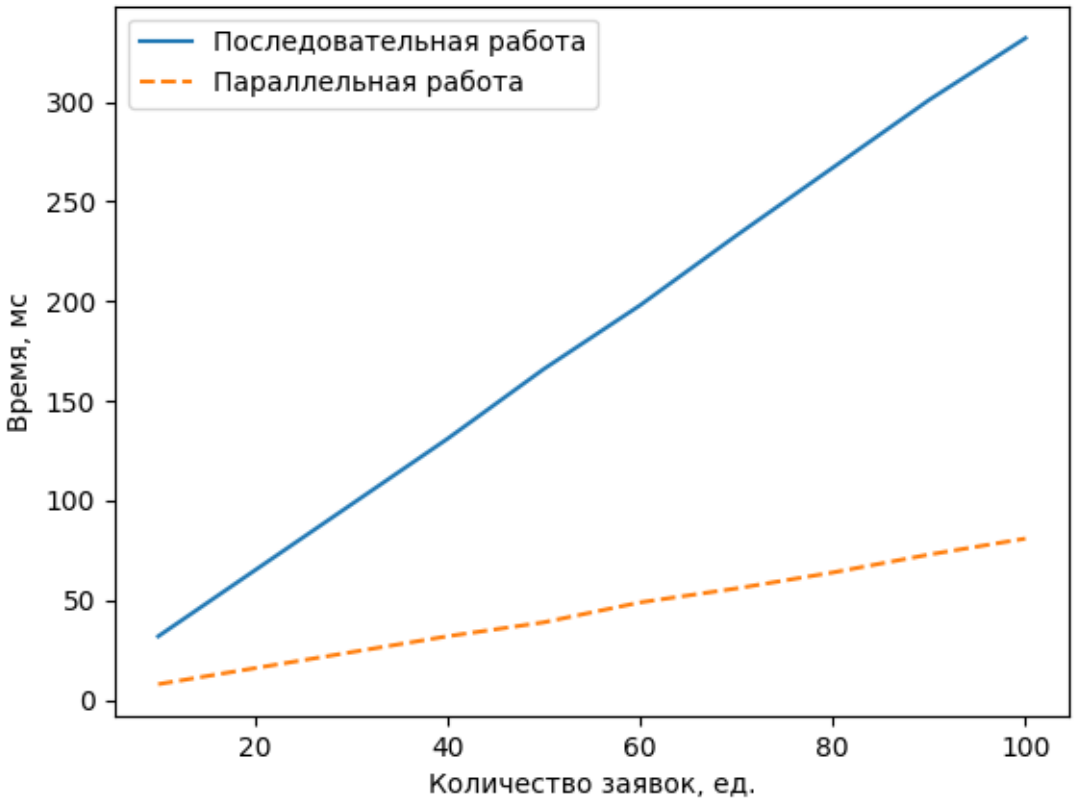

Рисунок 4.4 – Результаты замеров времени работы реализаций алгоритма с разным количеством потоков в зависимости от количества документов

Вывод

В результате эксперимента было получено, что использование конвейерной обработки способно сократить время обработки 100 заявок в 4.1 раза. В силу линейности графиков (рис. 4.4) можно сказать, что на достаточно большом количестве заявок выигрыш параллельной обработки над последовательной во времени в абсолютных единицах будет увеличиваться.

Заключение

Цель, которая была поставлена в начале лабораторной работы, была достигнута: применены принципы конвейерной обработки данных.

Кроме того были выполнены все поставленные задачи:

1) были описаны основы конвейерной обработки данных;
2) был описан алгоритм выделения наиболее информативных терминов набора документов на основе документной частоты;
3) был разработан и реализован программный продукт, позволяющий выполнять как последовательную, так и конвейерную реализацию озвученного алгоритма;
4) был проведен сравнительный анализ времени выполнения последовательной и конвейерной реализаций алгоритма;
5) был составлен отчет о выполненной лабораторной работе.

Список использованных источников

[1] Конвейерная обработка данных [Электронный ресурс]. Режим доступа: https://studref.com/636041/ekonomika/konveyernaya_obrabotka_dannyh (дата обращения: 14.12.2022).

[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/cpp/language (дата обращения: 10.11.2022).

[4] Класс thread [Электронный ресурс]. Режим доступа:https://learn.microsoft.com/ru-ru/cpp/standard-library/thread-class (дата обращения: 14.11.2022).

[5] Класс mutex [Электронный ресурс]. Режим доступа:https://learn.microsoft.com/ru-ru/cpp/standard-library/mutex-class-stl (дата обращения: 14.11.2022).

[6] Класс queue [Электронный ресурс]. Режим доступа:https://learn.microsoft.com/ru-ru/cpp/standard-library/queue-class (дата обращения: 14.11.2022).

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

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

[9] 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).