



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 работа №5

по курсу «Анализ Алгоритмов»

на тему: «Организация асинхронного взаимодействия потоков вычисления на
примере конвейерных вычислений»

Студент ИУ7-53Б
(Группа)

(Подпись, дата)

Лысцев Н. Д.
(И. О. Фамилия)

Преподаватель

(Подпись, дата)

Волкова Л. Л.
(И. О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Аналитический раздел	4
1.1 Конвейерная обработка данных	4
1.2 N-граммы	4
1.3 Последовательная версия алгоритма	5
1.4 Параллельная версия алгоритма	5
1.5 Описание алгоритма	6
2 Конструкторский раздел	7
2.1 Разработка алгоритмов	7
2.1.1 Параллельный алгоритм составления файла-словаря с N-граммами	7
2.1.2 Последовательный алгоритм работы стадий конвейера	9
2.1.3 Параллельный алгоритм работы стадий конвейера	10
3 Технологический раздел	16
3.1 Требования к программному обеспечению	16
3.2 Средства реализации	16
3.3 Сведения о модулях программы	17
3.4 Реализации алгоритмов	18
4 Исследовательский раздел	24
4.1 Технические характеристики	24
4.2 Демонстрация работы программы	24
4.3 Проведение исследования	25
ЗАКЛЮЧЕНИЕ	29
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	30

ВВЕДЕНИЕ

Использование параллельной обработки открывает новые способы для ускорения работы программ. Конвейерная обработка является одним из примеров, где использование принципов параллельности помогает ускорить обработку данных. Суть та же, что и при работе реальных конвейерных лент — материал (данное) поступает на обработку, после окончания обработки материал передается на место следующего обработчика, при этом предыдущий обработчик не ждет полного цикла обработки материала, а получает новый материал и работает с ним.

Целью данной лабораторной работы является описание параллельных конвейерных вычислений.

Для достижения поставленной цели необходимо решить следующие задачи:

- 1) изучить и описать организацию конвейерной обработки данных;
- 2) описать алгоритмы обработки данных, которые будут использоваться в текущей лабораторной работе;
- 3) определить средства программной реализации;
- 4) реализовать программу, выполняющую конвейерную обработку с количеством лент не менее трех в однопоточной и многопоточной реализаций;
- 5) сравнить и проанализировать реализации алгоритмов по затраченному времени.

1 Аналитический раздел

В данном разделе рассмотрена информация, касающаяся основ конвейерной обработки данных.

1.1 Конвейерная обработка данных

Конвейер — организация вычислений, при которой увеличивается количество выполняемых инструкций за единицу времени за счет использования принципов параллельности.

Конвейеризация (или 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. 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 увеличивает 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 [1].

1.2 N-граммы

Пусть задан некоторый конечный алфавит

$$V = w_i \tag{1.1}$$

где w_i — символ.

Языком $L(V)$ называют множество цепочек конечной длины из символов w_i . N-граммой на алфавите V (1.1) называют произвольную цепочку из $L(V)$ длиной N , например последовательность из N букв русского языка одного слова,

одной фразы, одного текста или, в более интересном случае, последовательность из грамматически допустимых описаний N подряд стоящих слов [2].

1.3 Последовательная версия алгоритма

Алгоритм составления файла словаря с количеством употреблений каждой N -граммы букв из одного слова в тексте на русском языке состоит из следующих шагов:

- 1) считывание текста в массив строк;
- 2) преобразование считанного текста (перевод букв в нижний регистр, удаление знаков препинания);
- 3) обработка каждой строки текста.

Обработка строки текста включает следующие шаги:

- 1) обработка каждого слова из строки текста;
- 2) выделение существующих в этом слове N -грамм;
- 3) увеличение количества выделенных N -грамм в словаре.

1.4 Параллельная версия алгоритма

В алгоритме составления файла словаря с количеством употреблений каждой N -граммы букв из одного слова в тексте на русском языке обработка строк текста происходит независимо, поэтому есть возможность произвести распараллеливание данных вычислений.

Для этого строки текста поровну распределяются между потоками. Каждый поток получает локальную копию словаря для N -грамм, производит вычисления над своим набором строк и после завершения работы всех потоков словари с количеством употреблений N -грамм каждого потока объединяются в один. Так как каждая строка массива передается в монопольное использование каждому потоку, не возникает конфликтов доступа к разделяемым ячейки памяти, следовательно, в использовании средства синхронизации в виде мьютекса нет необходимости.

1.5 Описание алгоритма

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

Заявка будет содержать:

- имя файла с исходным текстом на русском языке;
- имя выходного файла для словаря N -грамм;
- число N .
- временные отметки начала и конца выполнения каждой стадии обработки заявки.

В качестве операций, выполняющихся на конвейере, взяты следующие:

- 1) чтение текста на русском языке из файла;
- 2) выполнение параллельной версии алгоритма составления файла словаря с количеством употреблений каждой N -граммы букв из одного слова в тексте на русском языке в 2 потока;
- 3) логирование заявки в файл в формате: <имя_файла> + словарь N -грамм;

Вывод

В данном разделе было рассмотрено понятие конвейерной обработки, а также выбраны этапы обработки файлов, которые будут обрабатывать ленты конвейера. Также рассмотрено понятие N -грамм.

Программа будет получать на вход количество задач, число N , число строк в файле и длина каждой строки, а также выбор алгоритма: последовательный или конвейерный.

2 Конструкторский раздел

2.1 Разработка алгоритмов

2.1.1 Параллельный алгоритм составления файла-словаря с N-граммами

На рисунках 2.1 и 2.2 представлена схема параллельного алгоритма составления файла-словаря с N-граммами.

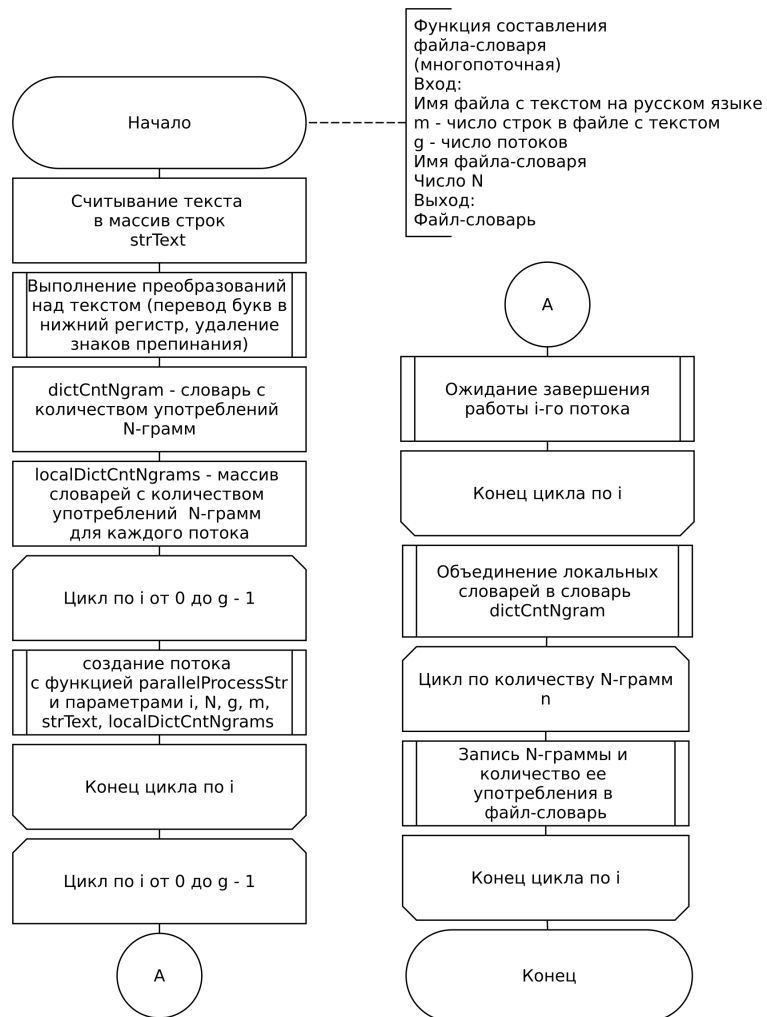


Рисунок 2.1 – Схема параллельного алгоритма составления файла-словаря с N-граммами

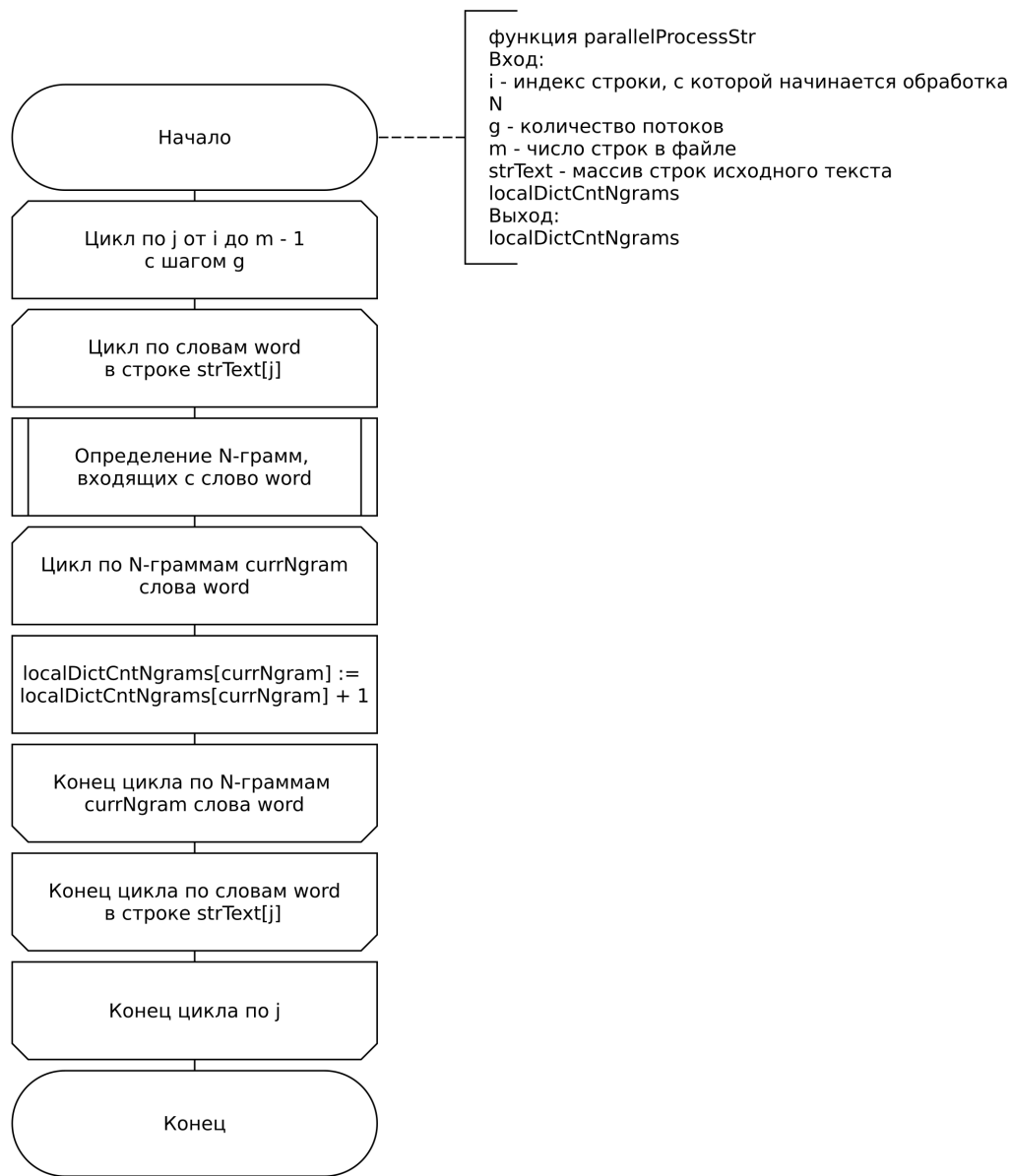


Рисунок 2.2 – Схема алгоритма функции для одного потока

2.1.2 Последовательный алгоритм работы стадий конвейера

На рисунке 2.3 представлен последовательный алгоритм работы стадий конвейера.

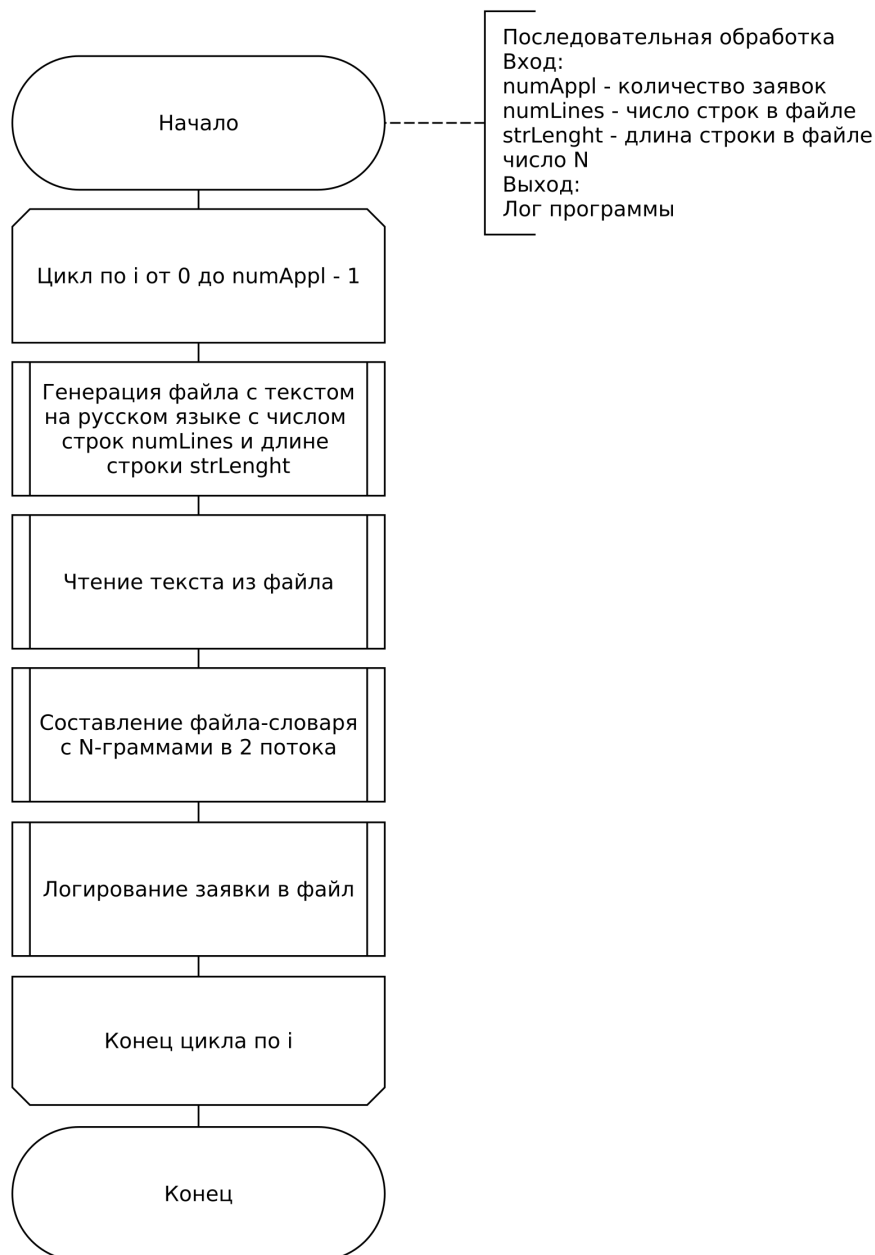


Рисунок 2.3 – Схема последовательного алгоритма работы стадий конвейера

2.1.3 Параллельный алгоритм работы стадий конвейера

Параллельная работа будет реализована с помощью добавления 3-х вспомогательных потоков, где каждый поток отвечает за свою стадию обработки.

Вспомогательному потоку в числе аргументов в качестве структуры будут переданы:

- имя файла с исходным текстом на русском языке;
- имя выходного файла для словаря N -грамм;
- число N .
- временные отметки начала и конца выполнения каждой стадии обработки заявки.

На рисунке 2.4 представлена схема алгоритма работы главного потока при параллельной работе стадий конвейера.

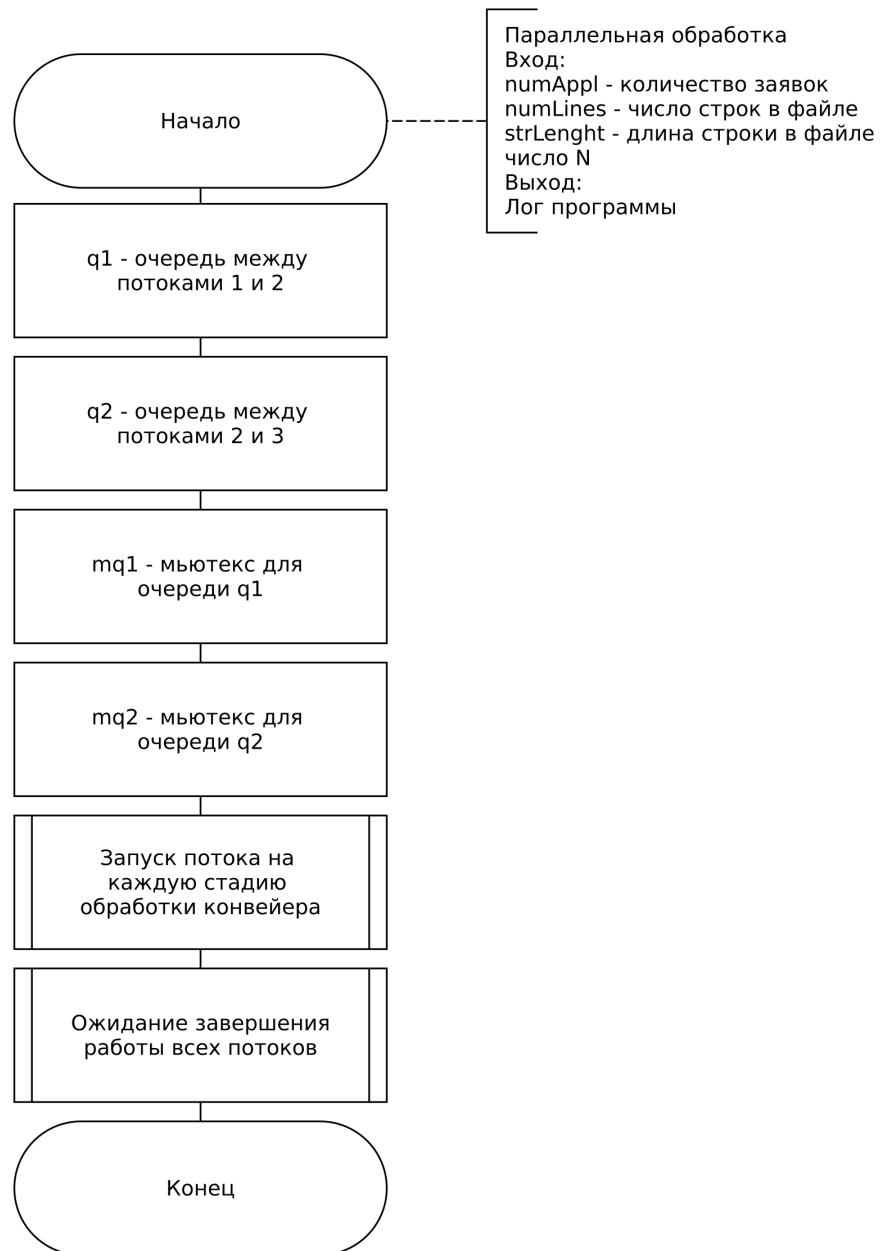


Рисунок 2.4 – Схема параллельного алгоритма работы стадий конвейера

На рисунках 2.5 – 2.7 представлены схемы алгоритмов каждого из обработчиков (потоков) при параллельной работе.

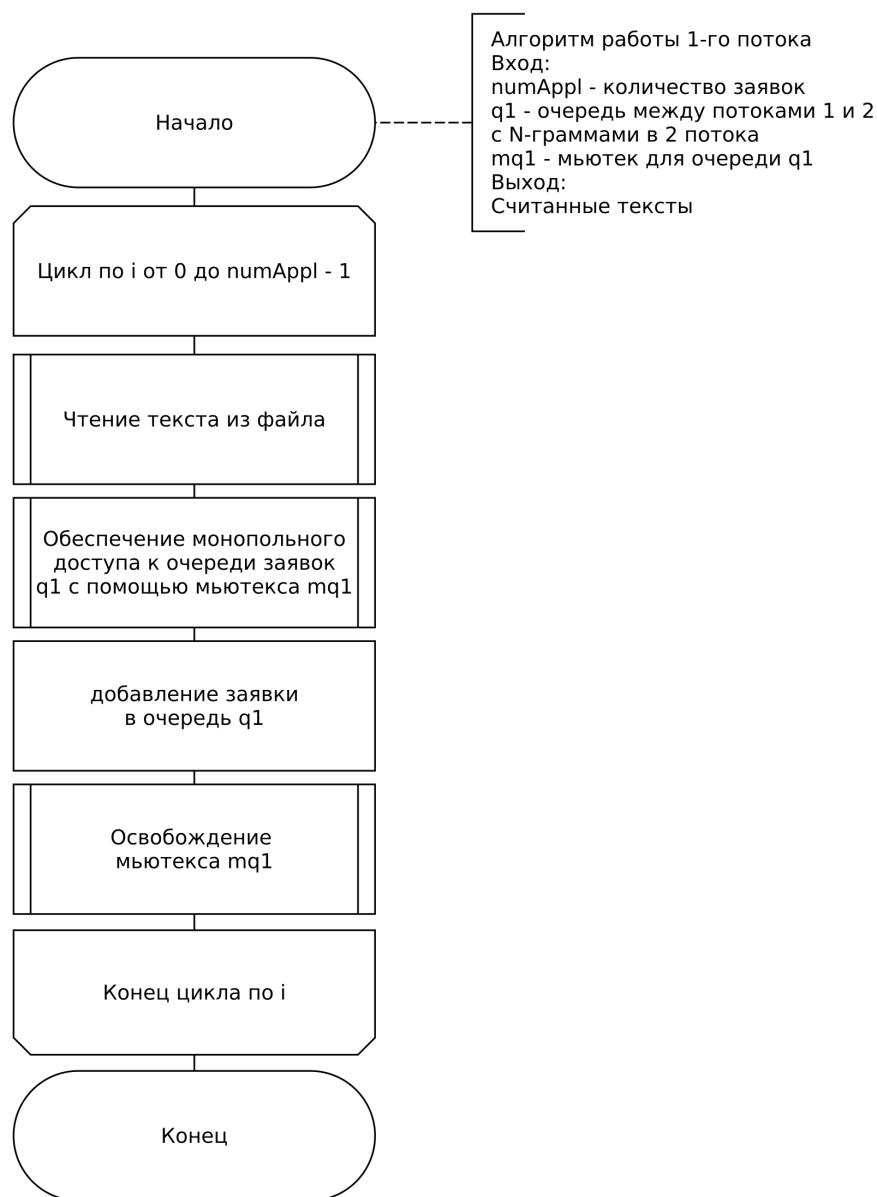


Рисунок 2.5 – Схема алгоритма потока, выполняющего чтение текстов из файлов

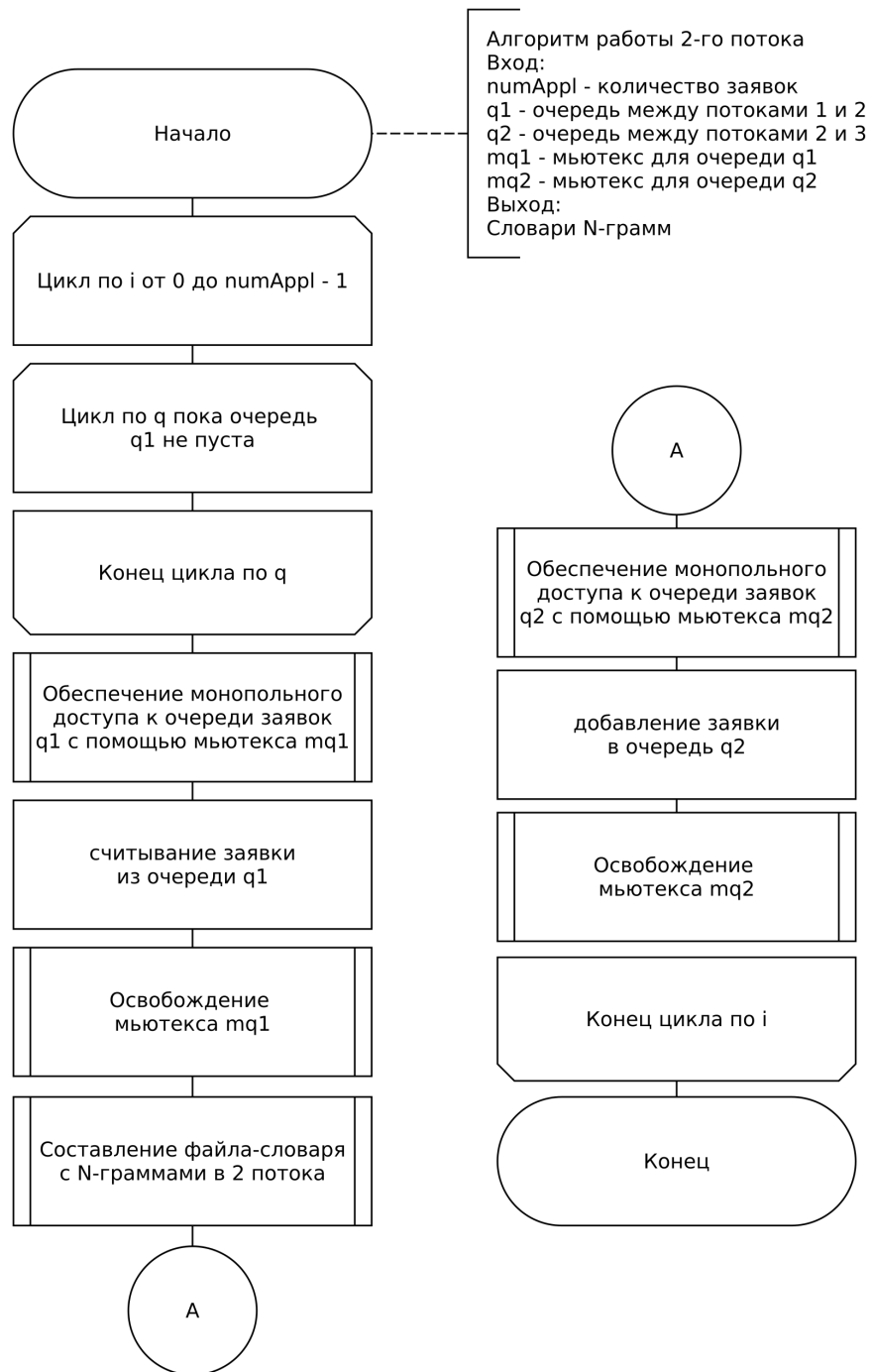


Рисунок 2.6 – Схема алгоритма потока, выполняющего составление файлов-словарей с N -граммами

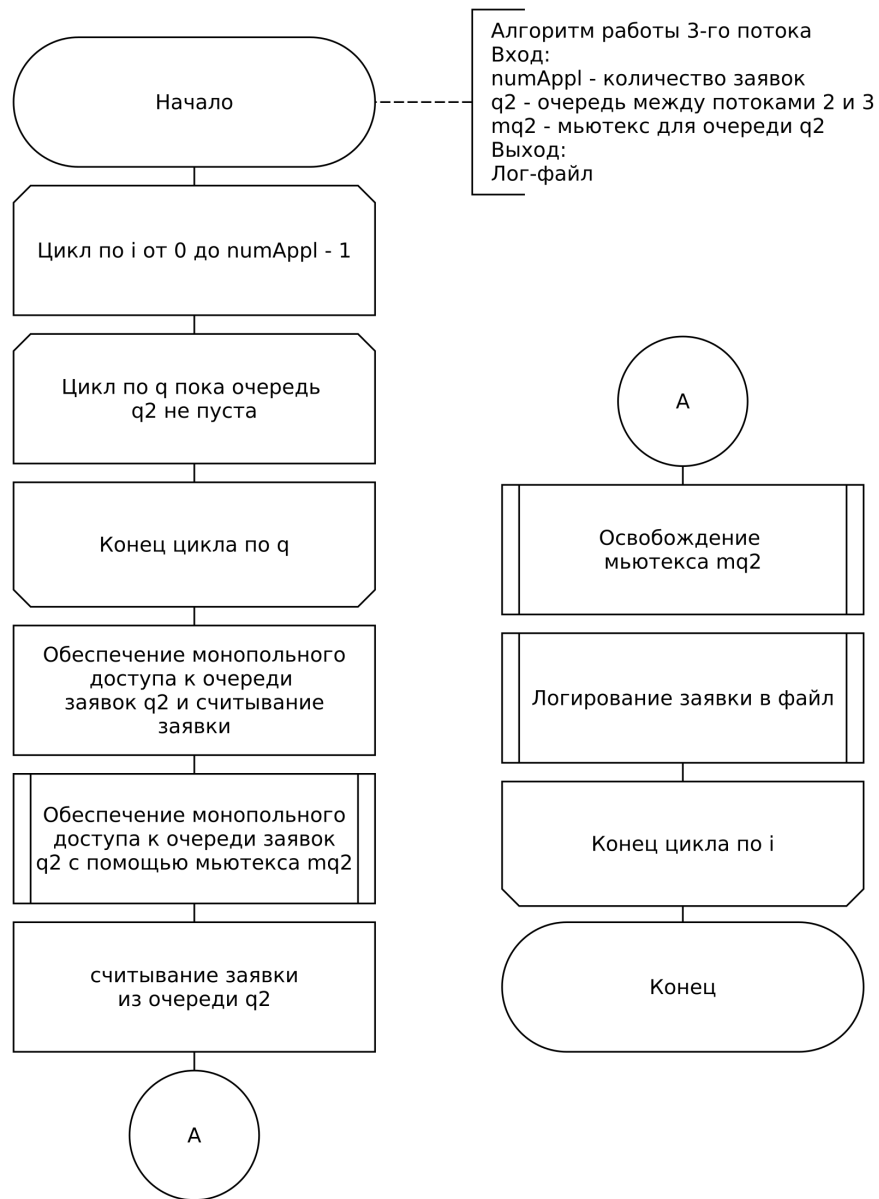


Рисунок 2.7 – Схема алгоритма потока, выполняющего логирование заявок в файл

Вывод

В данном разделе были представлены схемы последовательной и параллельной работы стадий конвейера.

3 Технологический раздел

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

3.1 Требования к программному обеспечению

К программе предъявляется ряд требований:

- наличие меню для выбора запускаемого режима работы конвейера — последовательного и параллельного;
- предоставление интерфейса для ввода количества заявок, числа строк в файлах, длины строки в файле и числа N ;
- формирование файла с логом работы конвейера, логирование событий обработки должно происходить после окончания работы, собственно, конвейера.

3.2 Средства реализации

В качестве языка программирования для этой лабораторной работы был выбран $C++$ [3] по следующим причинам:

- в $C++$ есть встроенный модуль *ctime*, предоставляющий необходимый функционал для замеров процессорного времени;
- в $C++$ есть встроенный модуль *thread* [4], предоставляющий необходимый интерфейс для работы с нативными потоками.

В качестве функции, которая будет осуществлять замеры процессорного времени, будет использована функция *clock_gettime* из встроенного модуля *ctime* [5].

3.3 Сведения о модулях программы

Программа состоит из семи модулей:

- 1) `lab_04_code.cpp` — модуль, хранящий реализации параллельной версии алгоритма составления файла словаря с количеством употреблений каждой N -граммы букв из одного слова в тексте на русском языке;
- 2) `conveyor.cpp` — модуль, хранящий реализации параллельной версии алгоритма работы конвейера;
- 3) `serial.cpp` — модуль, хранящий реализации последовательной версии алгоритма работы конвейера;
- 4) `processTime.cpp` — модуль, содержащий функцию для замера процессорного времени;
- 5) `timeMeasurements.cpp` — модуль, содержащий функции, позволяющие провести сравнительный анализ использования времени;
- 6) `main.cpp` — файл, содержащий точку входа в программу;
- 7) `task7` — модуль, содержащий набор скриптов для проведения замеров программы по времени и памяти и построения графиков по полученным данным.

3.4 Реализации алгоритмов

В листингах 3.1 и 3.2 представлена реализация последовательной конвейерной обработки.

Листинг 3.1 – Реализация последовательной конвейерной обработки (начало)

```
void serialSolution(const int numAppl, const int numLines, const
    int strLenght, const int N)
{
    for (int i = 0; i < numAppl; ++i)
    {
        std::string inputFilename = "data/text" +
            std::to_string(i) + ".txt";
        getRandomText(inputFilename, numLines, strLenght);
    }

    std::ofstream logFile("data/log.txt");

    if (!logFile.is_open())
    {
        std::wcerr << L"Ошибка открытия файла" << std::endl;
        return;
    }

    std::vector<serialAppl_t> vecAppl;

    for (int i = 0; i < numAppl; ++i)
    {
        serialAppl_t appl;

        appl.inputFilename = "data/text" + std::to_string(i) +
            ".txt";
        appl.outputFilename = "data/outputText" +
            std::to_string(i) + ".txt";
        appl.N = N;
        appl.numLines = numLines;
        appl.strLenght = strLenght;
```

Листинг 3.2 – Реализация последовательной конвейерной обработки (конец)

```
std::vector<std::wstring> vecTextStr;

clock_gettime(CLOCK_REALTIME, &appl.timeStartRead);
parallelVersion::readFile(appl.inputFilename, vecTextStr);
clock_gettime(CLOCK_REALTIME, &appl.timeEndRead);

clock_gettime(CLOCK_REALTIME, &appl.timeStartProcess);
parallelVersion::solution(vecTextStr, appl.outputFilename,
    N, 2);
clock_gettime(CLOCK_REALTIME, &appl.timeEndProcess);

clock_gettime(CLOCK_REALTIME, &appl.timeStartLog);
logFile << appl.inputFilename << ": " <<
    appl.outputFilename << std::endl;
clock_gettime(CLOCK_REALTIME, &appl.timeEndLog);

vecAppl.push_back(appl);
}

logFile.close();

printVec(vecAppl, "data/serial.txt");
}
```

В листингах 3.3 и 3.4 представлена реализация параллельной конвейерной обработки.

Листинг 3.3 – Реализация основного потока для конвейерной обработки, создающих вспомогательные потоки

```
void conveyorSolution(const int numAppl, const int numLines, const
    int strLenght, const int N)
{
    // генерирую файлы с текстами
    for (int i = 0; i < numAppl; ++i)
    {
        std::string inputFilename = "data/text" +
            std::to_string(i) + ".txt";
        getRandomText(inputFilename, numLines, strLenght);
    }

    std::queue<conveyorAppl_t> q1;
    std::queue<conveyorAppl_t> q2;
    std::vector<conveyorAppl_t> vec;

    std::thread thr1(thrReadFileFunc, numAppl, std::ref(q1));
    std::thread thr2(thrProcessFunc, numAppl, N, std::ref(q1),
        std::ref(q2));
    std::thread thr3(thrLogFunc, numAppl, std::ref(q2),
        std::ref(vec));

    thr1.join();
    thr2.join();
    thr3.join();

    printVec(vec, "data/conveyor.txt");
}
```

Листинг 3.4 – Реализация функции вспомогательного потока, считывающего тексты из файлов

```
void thrReadFileFunc(const int numAppl, std::queue<conveyorAppl_t>
    &q1)
{
    for (int i = 0; i < numAppl; ++i)
    {
        conveyorAppl_t appl;

        appl.inputFilename = "data/text" + std::to_string(i) +
            ".txt";
        appl.outputFilename = "data/outputText" +
            std::to_string(i) + ".txt";

        clock_gettime(CLOCK_REALTIME, &appl.timeStartRead);
        parallelVersion::readFile(appl.inputFilename,
            appl.vecTextStr);
        clock_gettime(CLOCK_REALTIME, &appl.timeEndRead);

        mq1.lock();
        q1.push(appl);
        mq1.unlock();
    }
}
```

Листинг 3.5 – Реализация функции вспомогательного потока, выполняющая параллельную версию алгоритма составления файла словаря с количеством употреблений каждой N -граммы букв из одного слова в тексте на русском языке в 2 потока (начало)

```
void thrProcessFunc(const int numAppl, const int N,
    std::queue<conveyorAppl_t> &q1, std::queue<conveyorAppl_t> &q2)
{
    for (int i = 0; i < numAppl; ++i)
    {
        while (q1.empty())
            ;
    }
}
```

Листинг 3.6 – Реализация функции вспомогательного потока, выполняющая параллельную версию алгоритма составления файла словаря с количеством употреблений каждой N -граммы букв из одного слова в тексте на русском языке в 2 потока (конец)

```
mq1.lock();
conveyorAppl_t appl = q1.front();
q1.pop();
mq1.unlock();

clock_gettime(CLOCK_REALTIME, &appl.timeStartProcess);
parallelVersion::solution(appl.vecTextStr,
    appl.outputFilename, N, 2);
clock_gettime(CLOCK_REALTIME, &appl.timeEndProcess);

mq2.lock();
q2.push(appl);
mq2.unlock();
}
}
```

Листинг 3.7 – Реализация функции вспомогательного потока, выполняющая логирование заявки в файл (начало)

```
void thrLogFunc(const int numAppl, std::queue<conveyorAppl_t> &q2,
    std::vector<conveyorAppl_t> &vec)
{
    std::ofstream logFile("data/log.txt");

    if (!logFile.is_open())
    {
        std::wcerr << L"Ошибка открытия файла" << std::endl;
        return;
    }

    for (int i = 0; i < numAppl; ++i)
    {
        while (q2.empty())
            ;
    }
}
```

Листинг 3.8 – Реализация функции вспомогательного потока, выполняющая логирование заявки в файл (конец)

```
mq2.lock();
conveyorAppl_t appl = q2.front();
q2.pop();
mq2.unlock();

clock_gettime(CLOCK_REALTIME, &appl.timeStartLog);
logFile << appl.inputFilename << ": " <<
    appl.outputFilename << std::endl;
clock_gettime(CLOCK_REALTIME, &appl.timeEndLog);

vec.push_back(appl);
}

logFile.close();
}
```

Вывод

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

4 Исследовательский раздел

В данном разделе будут проведены сравнения реализаций алгоритмов сортировки по времени работы и по затрачиваемой памяти.

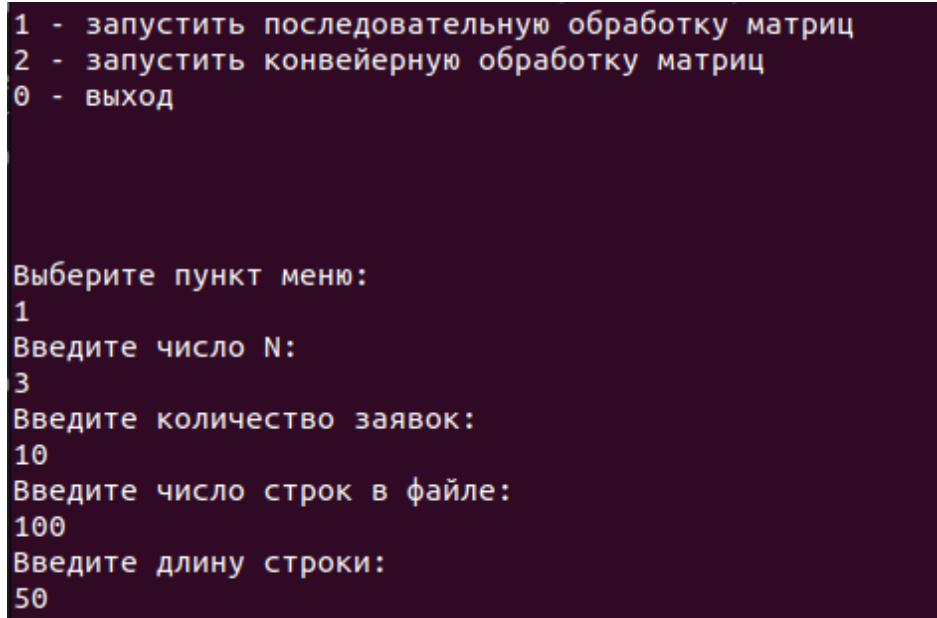
4.1 Технические характеристики

Технические характеристики устройства, на котором проводились исследования:

- операционная система: Ubuntu 22.04.3 LTS x86_64 [6];
- оперативная память: 16 Гб;
- процессор: 11th Gen Intel® Core™ i7-1185G7 @ 3.00 ГГц × 8, 4 физических ядра, 8 логических ядер.

4.2 Демонстрация работы программы

На рисунке 4.1 представлен пример результата работы программы.



```
1 - запустить последовательную обработку матриц
2 - запустить конвейерную обработку матриц
0 - выход

Выберите пункт меню:
1
Введите число N:
3
Введите количество заявок:
10
Введите число строк в файле:
100
Введите длину строки:
50
```

Рисунок 4.1 – Пример работы программы

На рисунке 4.2 представлен пример фрагмента лог-файла работы конвейера.

```
Заявка 0: start read      0 ns
Заявка 0: end read       335 397 ns
Заявка 0: start process  390 945 ns
Заявка 0: end process    14 391 493 ns
Заявка 0: start log      14 468 730 ns
Заявка 0: end log       14 520 901 ns
Заявка 1: start read     354 417 ns
Заявка 1: end read      579 622 ns
Заявка 1: start process  14 456 470 ns
Заявка 1: end process   25 713 219 ns
Заявка 1: start log     25 790 191 ns
Заявка 1: end log      25 797 074 ns
Заявка 2: start read     594 158 ns
Заявка 2: end read     852 065 ns
Заявка 2: start process  25 784 590 ns
Заявка 2: end process   36 570 799 ns
Заявка 2: start log     36 645 701 ns
Заявка 2: end log      36 650 183 ns
Заявка 3: start read    884 739 ns
Заявка 3: end read     1 529 707 ns
Заявка 3: start process  36 646 100 ns
Заявка 3: end process   48 322 275 ns
Заявка 3: start log     48 405 128 ns
Заявка 3: end log      48 408 658 ns
Заявка 4: start read    1 560 700 ns
```

Рисунок 4.2 – Пример фрагмента лог-файла работы конвейера

4.3 Проведение исследования

Цель исследования

Целью исследования является проведение сравнительного анализа времени работы последовательного и параллельного алгоритма работы конвейера от числа поступающих заявок.

Наборы варьируемых и фиксированных параметров

Замеры времени проводились для числа заявок, равному 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.

В качестве фиксированного параметра было выбрано число N , равное 3, число строк в файле, равное 100 и длина строки, равная 100 символам.

Замеры времени для каждого размера 1000 раз. Время работы алгоритмов измерялось с использованием функции *clock_gettime* из встроенного модуля *ctime* [5].

Результаты первого исследования

Таблица 4.1 – Замер времени для конвейера с числом заявок от 10 до 100

Число заявок, единицы	Время, мкс	
	Последовательная версия	Параллельная версия
10	158 528.300	182 195.500
20	295 954.100	327 198.500
30	522 347.500	597 439.000
40	768 368.400	804 989.700
50	971 115.000	914 300.400
60	1 084 875.000	1 209 416.000
70	1 370 885.000	1 380 609.000
80	1 473 052.000	1 378 848.000
90	1 613 111.000	1 691 645.000
100	1 871 918.000	2 125 750.000

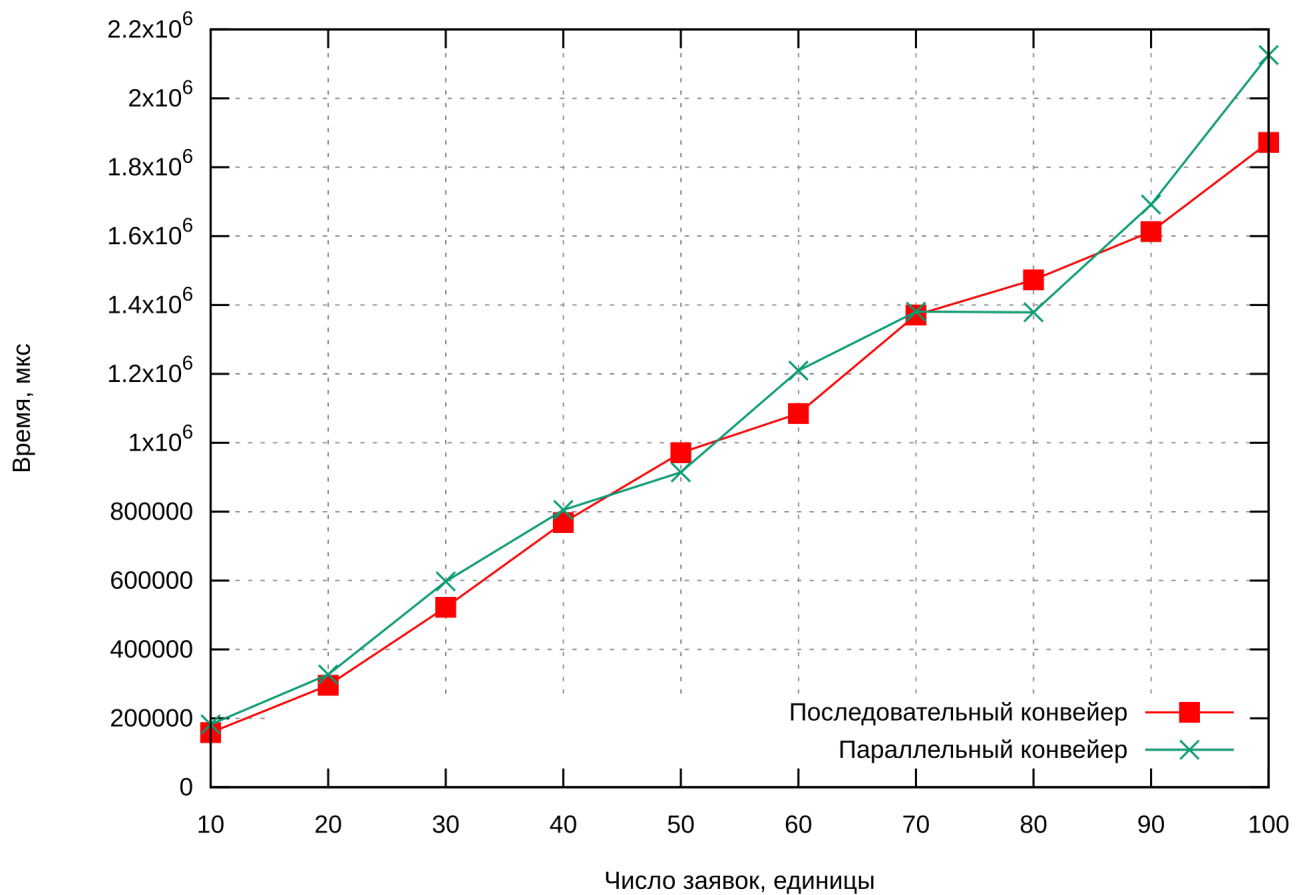


Рисунок 4.3 – Результаты замеров времени для конвейера с числом заявок от 10 до 100

Из полученных данных следует, что результаты использования конвейерной обработки не совпали с ожидаемыми: начиная с числа заявок, равного 90, наблюдается увеличение времени работы конвейера по сравнению с последовательным алгоритмом обработки заявок (на 90 заявках параллельный конвейер работает в 1.05 медленнее последовательного, а на 100 заявках параллельный конвейер работает уже в 1.14 медленнее последовательного). Вероятно, это связано с тем, что значительная часть работы уходит на создание и обслуживание вспомогательных потоков в конвейерной обработке.

Вывод

В результате исследования было установлено, что использование конвейерной обработки для обработки заявок оказалось менее эффективно, чем последовательная обработка заявок: начиная с числа заявок, равного 90, наблюдается увеличение времени работы конвейера по сравнению с последовательным алгоритмом обработки заявок (на 90 заявках параллельный конвейер работает в 1.05 медленнее последовательного, а на 100 заявках параллельный конвейер работает уже в 1.14 медленнее последовательного). Это связано с тем, что значительная часть работы уходит на создание и обслуживание вспомогательных потоков в конвейерной обработке.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были решены следующие задачи:

- 1) изучена и описана организация конвейерной обработки данных;
- 2) описаны алгоритмы обработки данных, которые использовались в текущей лабораторной работе;
- 3) определены средства программной реализации;
- 4) реализована программа, выполняющая конвейерную обработку с количеством лент не менее трех в однопоточной и многопоточной реализаций;
- 5) сравнены и проанализированы реализации алгоритмов по затраченному времени.

Цель данной лабораторной работы, а именно описание параллельных конвейерных вычислений, также была достигнута.

Для рассматриваемого алгоритма использование конвейерной обработки для обработки заявок приводит лишь к увеличению времени работы программы по сравнению с последовательной обработкой заявок: начиная с числа заявок, равного 90, наблюдается увеличение времени работы конвейера по сравнению с последовательным алгоритмом обработки заявок (на 90 заявках параллельный конвейер работает в 1.05 медленнее последовательного, а на 100 заявках параллельный конвейер работает уже в 1.14 медленнее последовательного).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Конвейерная обработка данных [Электронный ресурс]. — Режим доступа: <https://studfile.net/preview/1083252/page:25/> (дата обращения: 01.02.2024).
2. N-граммы в лингвистике [Электронный ресурс]. — Режим доступа: <https://cyberleninka.ru/article/n/n-grammy-v-lingvistike> (дата обращения: 27.01.2024).
3. Справочник по языку C++ [Электронный ресурс]. — Режим доступа: <https://learn.microsoft.com/ru-ru/cpp/cpp/cpp-language-reference?view=msvc-170> (дата обращения: 28.09.2022).
4. `std::thread` [Электронный ресурс]. — Режим доступа: <https://en.cppreference.com/w/cpp/thread/thread> (дата обращения: 27.01.2024).
5. `clock_getres` [Электронный ресурс]. — Режим доступа: https://pubs.opengroup.org/onlinepubs/9699919799/functions/clock_getres.html (дата обращения: 28.09.2022).
6. Ubuntu 22.04.3 LTS (Jammy Jellyfish) [Электронный ресурс]. — Режим доступа: <https://releases.ubuntu.com/22.04/> (дата обращения: 28.09.2022).