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»

Отчёт по лабораторной работе №5 по дисциплине "Анализ алгоритмов"

Тема Конвейерные вычисления

Студент Мицевич М. Д.

Группа ИУ7-51Б

Преподаватели Волкова Л.Л.

Москва — 2021 г.

Оглавление

Введение **3**

1 Аналитическая часть **4**
1.1 Конвейерная обработка данных 4
1.2 Описание задачи 4

2 Конструкторская часть **5**
2.1 Разработка алгоритмов 5
2.2 Используемые структуры данных 7

3 Технологическая часть **8**
3.1 Средства реализации 8
3.2 Реализация алгоритмов 8

4 Исследовательская часть **12**
4.1 Технические характеристики ЭВМ 12
4.2 Время выполнения алгоритмов 12

Заключение **16**

Литература **17**

Введение

При обработке данных могут возникать ситуации, когда один набор данных необходимо обработать последовательно несколькими алгоритмами. В таком случае удобно использовать идею конвейерной обработки. Конвейер - механизм организации труда, при котором производство изделия разбивается на стадии, и конкретные работники закрепляются за своими стадиями. Таким образом, обработка делится на линии конвейера. При этом каждая следующая линия обрабатывает данные только тогда, когда предыдущая завершила свою работу над теми же данными.

Здесь также можно заметить, что обработка данных на разных линиях конвейера может быть выполнена параллельно, что значительно улучшит производительность конвейра за счёт уменьшения времени простоя линий.

Целью данной работы является изучение и реализация параллельной и линейной реализаций конвейерной обработки данных. Для достижения поставленной цели необходимо решить следующие задачи:

- изучить конвейерную обработку данных;
- разработать метод конвейерной обработки данных на примере приготовления печенья;
- реализовать систему конвейерных вычислений;
- сравнить параллельную и последовательную реализации конвейерных вычислений на основе собранной статистики.

1 | Аналитическая часть

В данном разделе представлены теоретические сведения о рассматриваемых алгоритмах.

1.1 Конвейерная обработка данных

Конвейер - это механизм, который позволяет обрабатывать однотипные заявки поэтапно. Поэтому конвейер представляет собой некоторое количество лент, на каждой из которых выполняется одинаковая работа над поступающими заявками. В терминах программирования ленты конвейера представлены функциями-обработчиками, выполняющими над неким набором данных операции и передающими их на следующую ленту конвейера. Очевидно, что моделирование конвейерной обработки хорошо сочетается с технологией многопоточного программирования, так как под каждую ленту конвейера может быть выделен отдельный поток.

1.2 Описание задачи

Для моделирования системы конвейерной обработки данных был выбран процесс приготовления печенья в производственных масштабах, состоящий из трех этапов:

- подсчёт массы куриных яиц в мг (вычисление n-ого числа Фибоначчи);
- подсчёт массы масла в мг (возведение в степень)
- подсчёт массы муки в мг (факториал числа).

Вывод

В данном разделе были рассмотренны особенности построения конвейерных вычислений.

2 | Конструкторская часть

В данном разделе представлены схемы рассматриваемых алгоритмов.

2.1 Разработка алгоритмов

На рисунке 2.1 приведена общая схема организации конвейерных вычислений для каждой заявки.

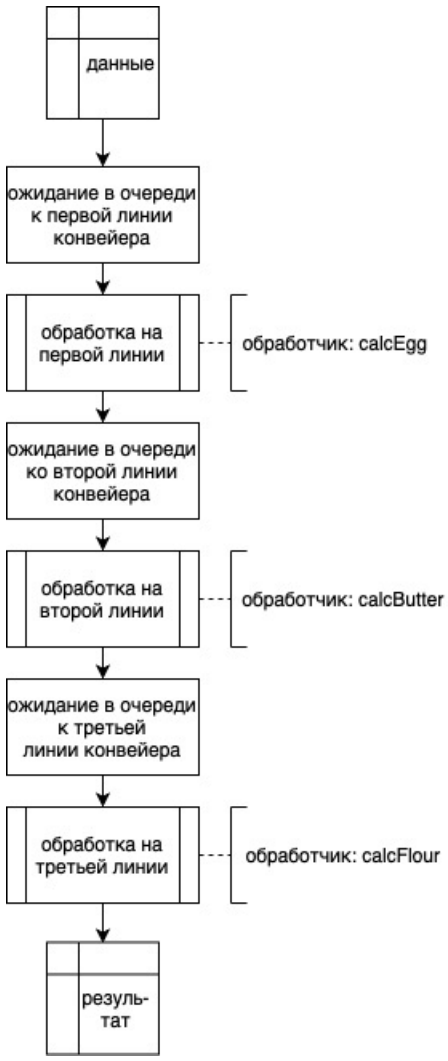

Рис. 2.1: Общая схема организации конвейерных вычислений.

На рисунке 2.2 приведена схема главного потока для параллельной обработки заявок.

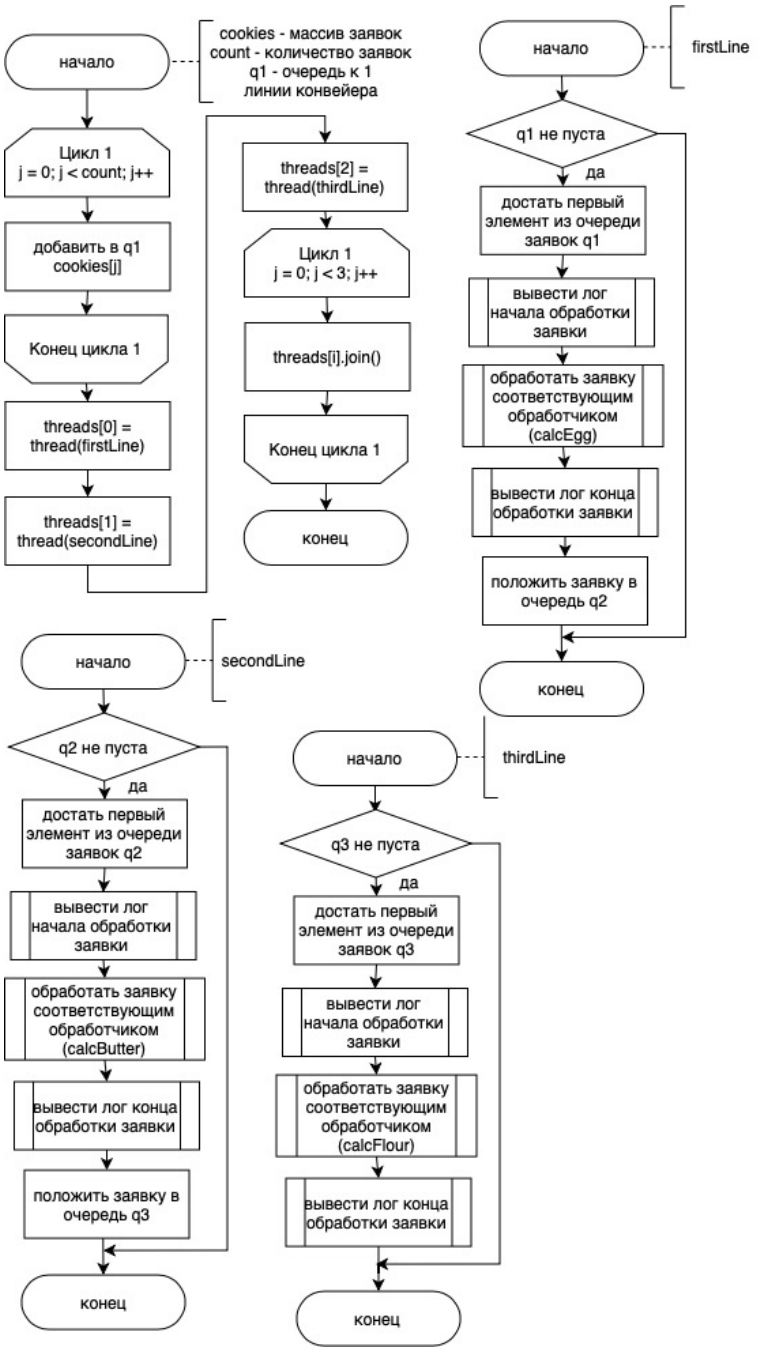

Рис. 2.2: Схема главного потока для параллельной обработки заявок

На рисунке 2.3 приведены схемы каждого типа вычислений конвейера.

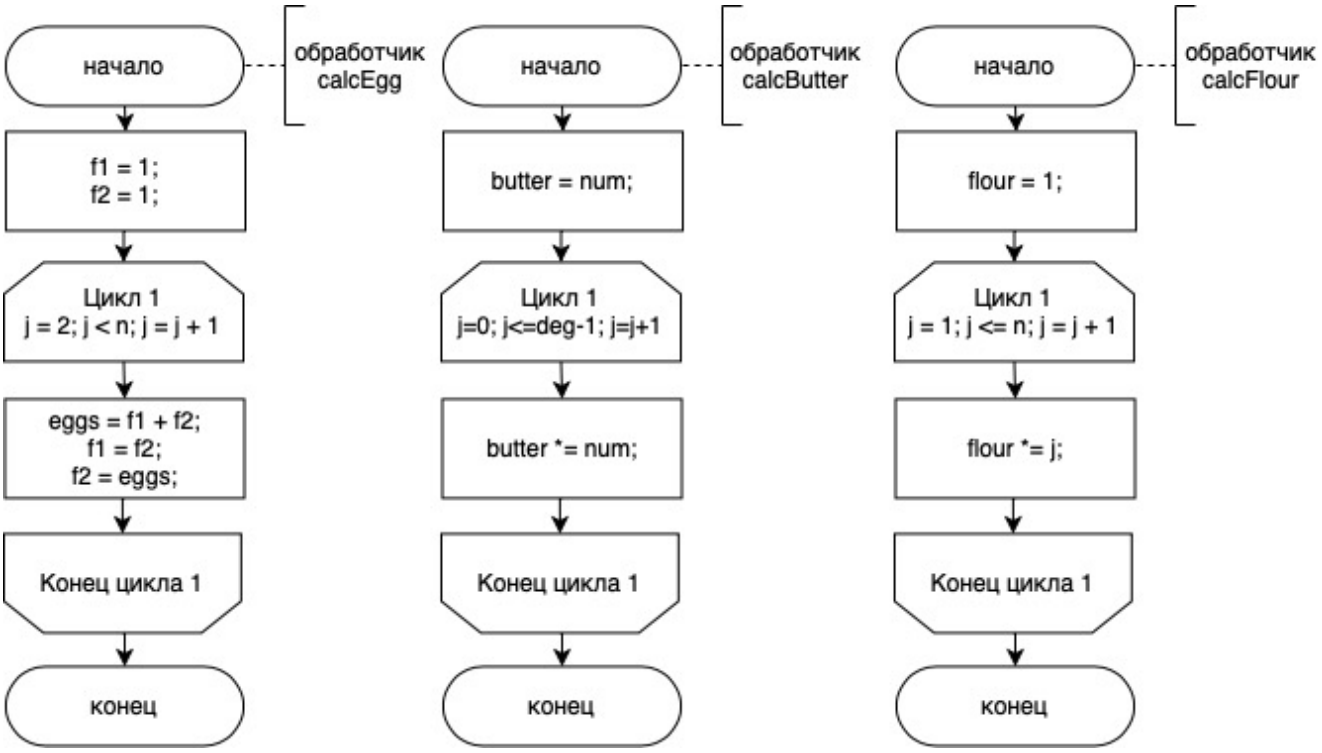

Рис. 2.3: Схемы трёх типов вычислений конвейера

2.2 Используемые структуры данных

Для организации потокобезопасных очередей было решено использовать обёртку над стандартным типом данных очередь. Такая очередь обеспечивает монопольное владение ресурсом каждым потоком, что не позволяет привести к ситуации гонки. Так, каждый вызов методов очереди блокируется мьютексом, который позволяет защитить критические участка кода своей блокировкой и запрещает доступ к своей области памяти, пока заблокирован.

Вывод

На основе анализа принципа работы конвейера была разработана общая схема организации конвейерных вычислений и схемы вычислений на каждой ленте конвейера в соответствии с выбранным для моделирования процессом. Кроме того, были выбраны используемые структуры данных.

3 | Технологическая часть

В данном разделе приведены средства реализации и листинги кода.

3.1 Средства реализации

Для реализации был выбран язык программирования C++. Данный выбор обусловлен наличием инструментов для создания и эффективной работы с потоками.

3.2 Реализация алгоритмов

В листингах 3.1 и 3.2 приведены однопоточная и многопоточная реализации конвейерных вычислений, в листинге 3.3 - реализация потокобезопасной очереди, а в листинге 3.4 - реализация обработчиков, представляющих этапы конвейера.

Листинг 3.1: Однопоточная реализация конвейера

```
1
2 void Conveyor::run_seq(size_t count) {
3     for (size_t i = 0; i < count; i++) {
4         std::shared_ptr<Cookies> c(new Cookies);
5         c->calcEgg(1000000);
6         c->calcButter(2, 1000000);
7         c->calcFlour(1000000);
8     }
9 }
```

Листинг 3.2: Многопоточная реализация конвейера

```
1  void Conveyor::run_par(size_t count) {
2      for (size_t i = 0; i < count; i++) {
3          std::shared_ptr<Cookies> p(new Cookies);
4          cookies.push_back(p);
5          q1.push(p);
6      }
7      this->threads[0] = std::thread(&Conveyor::calcEgg, this);
8      this->threads[1] = std::thread(&Conveyor::calcButter, this);
9      this->threads[2] = std::thread(&Conveyor::calcFlour, this);
10     for (int i = 0; i < 3; i++) {
11         this->threads[i].join();
12     }
```

```
}
void Conveyor::calcEgg()
{
    size_t task_num = 1;
    while (!this->q1.empty()) {
        std::shared_ptr<Cookies> c = q1.front();
        log_current_event(task_num, "Part 1 | Start");
        c->calcEgg(1000000);
        log_current_event(task_num, "Part 1 | End");
        q2.push(c);
        q1.pop();
        task_num++;
    }

}
void Conveyor::calcButter()
{
    size_t task_num = 1;
    do {
        if (!this->q2.empty()) {
            std::shared_ptr<Cookies> c = q2.front();
            log_current_event(task_num, "Part 2 | Start");
            c->calcButter(2, 1000000);
            log_current_event(task_num, "Part 2 | End");
            q3.push(c);
            q2.pop();
            task_num++;
        }
    } while(!q1.empty() || !q2.empty());
}
void Conveyor::calcFlour()
{
    size_t task_num = 1;
    do {
        if (!this->q3.empty()) {
            std::shared_ptr<Cookies> c = q3.front();
            log_current_event(task_num, "Part 3 | Start");
            c->calcFlour(1000000);
            log_current_event(task_num, "Part 3 | End");
            q3.pop();
            task_num++;
        }
    } while (!q1.empty() || !q2.empty() || !q3.empty());
}
```

Листинг 3.3: Реализация потокобезопасной очереди

```
std::mutex mtx;
template <typename type>
void SafeQueue<type>::push(type val)
{
    mtx.lock();
    q.push(val);
    mtx.unlock();
}
template <typename type>
void SafeQueue<type>::pop()
{
    mtx.lock();
    q.pop();
    mtx.unlock();
}
template <typename type>
bool SafeQueue<type>::empty()
{
    bool r;
    mtx.lock();
    r = q.empty();
    mtx.unlock();
    return r;
}
template <typename type>
type SafeQueue<type>::front()
{
    type val;
    mtx.lock();
    val = q.front();
    mtx.unlock();
    return val;
}
```

Листинг 3.4: Обработчики линий конвейера

```
void Cookies::calcEgg(int n) // fib
{
    int f1 = 1, f2 = 1;

    for (int i = 2; i < n; i++)
    {
        this->eggs = f1 + f2;
        f1 = f2;
        f2 = this->eggs;
    }
}
void Cookies::calcButter(int num, int deg)  // num ^ degree
{
    this->butter = num;
    for (int i = 0; i < deg - 1; i++)
        this->butter *= num;
}
void Cookies::calcFlour(long n)  // factorial
{
    this->flour = 1;
    for (int i = 1; i <= n; i++)
        this->flour *= i;
}
```

Вывод

В данном разделе была реализована конвейерная обработка данных на примере моделировани процесса приготовления печенья, кроме того, были выбраны средства разработки.

4 | Исследовательская часть

В данном разделе приведен анализ характеристик разработанного ПО и примеры работы ПО.

4.1 Технические характеристики ЭВМ

Все нижепреведенные замеры времени проведены на процессоре: Intel Core i5, 1,4 GHz, четырёхъядерный, количество логических ядер - 8. Время работы алгоритмов было замерено с помощью chrono [1].

4.2 Время выполнения алгоритмов

На рисунке 4.1 приведено сравнение времени выполнения параллельной и последовательной обработки данных в зависимости от количества входных задач. При этом предполгаается, что на лентах конвейера заявки обрабатываются примерно за одинаковое время. В таблице 4.2 приведена статистика выполнения заявок в зависимости от количества заявок: указано среднее время выполнения заявки (время, которое заявка занимала у всех линий конвейера), среднее время ожидания в очереди и среднее время пребывания в системе. В листинге 4.1 приведён лог параллельной обработки восьми заявок.

Таблица 4.1: Таблица времени выполнения параллельной обработки данных (в мс) с печатью лога

Количество задач	t в очереди	t в системе	t обработки
100	213.990	224.430	10.440
200	442.710	453.185	10.475
300	667.347	677.813	10.467
400	907.885	918.395	10.510
500	957.998	967.982	9.984
600	1370.842	1381.335	10.493
700	1587.126	1597.559	10.433
800	1802.399	1812.965	10.566
900	2022.898	2033.367	10.469
1000	2275.780	2286.326	10.546

Таблица 4.2: Таблица времени выполнения параллельной обработки данных (в мс) без печати лога

Количество задач	t в очереди	t в системе	t обработки
100	201.056	213.006	10.050
200	435.410	445.485	10.075
300	654.457	664.323	9.967
400	903.885	913.395	10.510
500	949.564	959.784	9.200
600	1350.421	1361.631	10.712
700	1564.387	1574.547	10.123
800	1782.399	1792.965	10.566
900	1997.898	2007.367	10.469
1000	2275.780	2286.326	10.546

Листинг 4.1: Лог параллельной обработки восьми заявок

```
Task 1 | Part 1 | Start | 16:10:13.547
Task 1 | Part 1 | End | 16:10:13.828
Task 2 | Part 1 | Start | 16:10:13.828
Task 1 | Part 2 | Start | 16:10:13.828
Task 1 | Part 2 | End | 16:10:14. 20
Task 1 | Part 3 | Start | 16:10:14. 20
Task 2 | Part 1 | End | 16:10:14. 84
Task 3 | Part 1 | Start | 16:10:14. 84
Task 2 | Part 2 | Start | 16:10:14. 84
Task 1 | Part 3 | End | 16:10:14.300
Task 2 | Part 2 | End | 16:10:14.699
Task 2 | Part 3 | Start | 16:10:14.699
Task 3 | Part 1 | End | 16:10:15.233
Task 4 | Part 1 | Start | 16:10:15.233
Task 3 | Part 2 | Start | 16:10:15.233
Task 2 | Part 3 | End | 16:10:15.275
Task 3 | Part 2 | End | 16:10:15.457
Task 3 | Part 3 | Start | 16:10:15.457
Task 4 | Part 1 | End | 16:10:15.497
Task 5 | Part 1 | Start | 16:10:15.497
Task 4 | Part 2 | Start | 16:10:15.497
Task 4 | Part 2 | End | 16:10:15.692
Task 3 | Part 3 | End | 16:10:15.731
Task 4 | Part 3 | Start | 16:10:15.731
Task 5 | Part 1 | End | 16:10:15.754
Task 6 | Part 1 | Start | 16:10:15.754
Task 5 | Part 2 | Start | 16:10:15.754
Task 5 | Part 2 | End | 16:10:15.951
Task 4 | Part 3 | End | 16:10:16.  0
Task 5 | Part 3 | Start | 16:10:16.  0
```

```
Task 6 | Part 1 | End | 16:10:16.  2
Task 7 | Part 1 | Start | 16:10:16.  2
Task 6 | Part 2 | Start | 16:10:16.  2
Task 6 | Part 2 | End | 16:10:16.202
Task 7 | Part 1 | End | 16:10:16.265
Task 8 | Part 1 | Start | 16:10:16.265
Task 7 | Part 2 | Start | 16:10:16.265
Task 5 | Part 3 | End | 16:10:16.278
Task 6 | Part 3 | Start | 16:10:16.278
Task 7 | Part 2 | End | 16:10:16.446
Task 8 | Part 1 | End | 16:10:16.507
Task 8 | Part 2 | Start | 16:10:16.507
Task 6 | Part 3 | End | 16:10:16.539
Task 7 | Part 3 | Start | 16:10:16.539
Task 8 | Part 2 | End | 16:10:16.687
Task 7 | Part 3 | End | 16:10:16.802
Task 8 | Part 3 | Start | 16:10:16.802
Task 8 | Part 3 | End | 16:10:17. 66
```

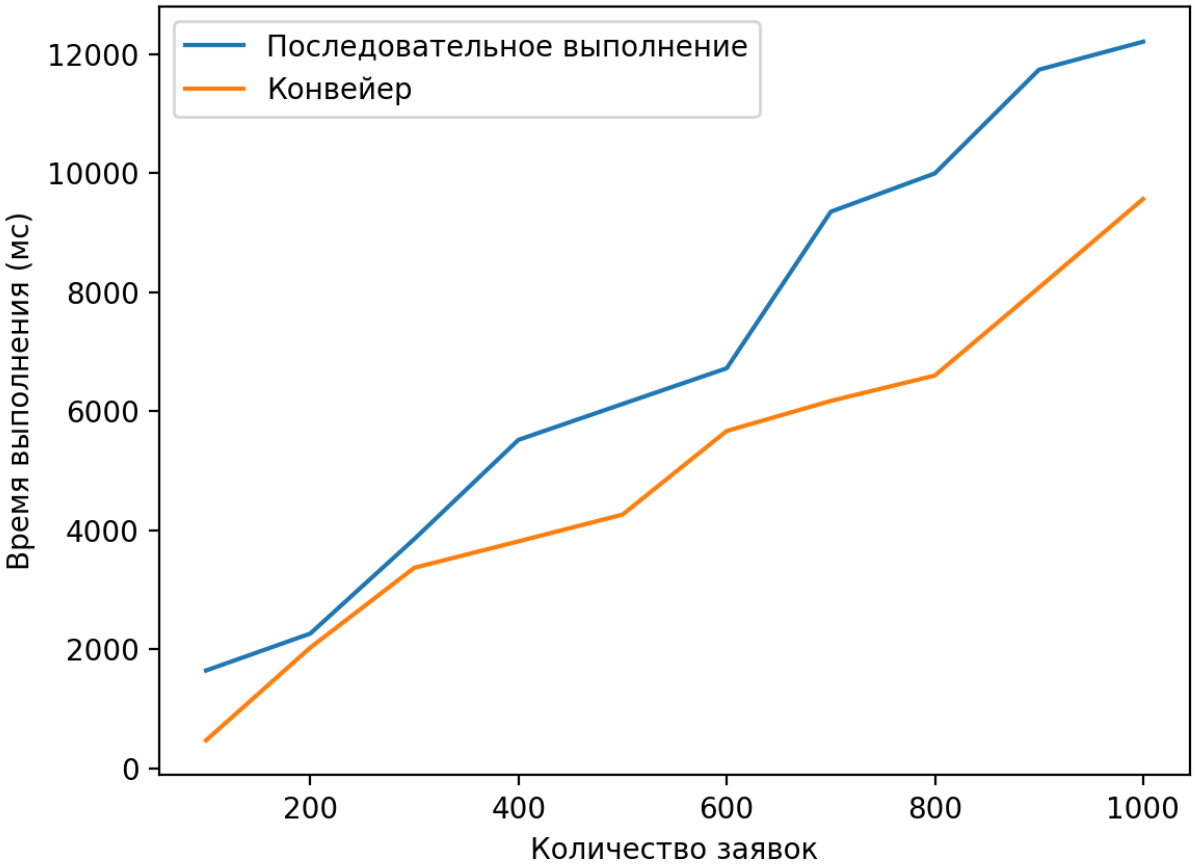

Рис. 4.1: Сравнение времени выполнения параллельной и последовательной обработки данных (в мс).

Вывод

В результате исследования было выяснено, что многопотчная реализация конвейера работает быстрее, чем линейная. Кроме того, с увеличением количества заявок растёт и время ожидания в очереди к линиям конвейера.

Заключение

В рамках данной лабораторной работы были решены следующие задачи:

- изучена конвейерная обработка данных;
- разработан метод конвейерной обработки данных на примере приготовления печенья;
- реализована система конвейерных вычислений;
- было проведено сравнение параллельной и линейной реализаций конвейерных вычислений на основе собранной статистики.

Параллельные конвейерные вычисления позволяют организовать непрерывную обработку данных, что позволяет выиграть время в задачах, где требуется обработка больших объемов данных за малый промежуток времени.

Поставленная цель была достигнута.

Литература

1. Стандартный набор функций для получения значения времени [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/cpp/chrono. Дата обращения: 16.11.2021