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 Конвейерная обработка данных

Студент Жабин Д.В.

Группа ИУ7-54Б

Преподаватель Волкова Л.Л.

Москва, 2021 г.

Содержание

Введение

Конвейер — способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности (увеличения числа инструкций, выполняемых в единицу времени — эксплуатация параллелизма на уровне инструкций), технология, используемая при разработке компьютеров и других цифровых электронных устройств.

Идея заключается в параллельном выполнении нескольких инструкций процессора. Сложные инструкции процессора представляются в виде последовательности более простых стадий. Вместо выполнения инструкций последовательно (ожидания завершения конца одной инструкции и перехода к следующей), следующая инструкция может выполняться через несколько стадий выполнения первой инструкции. Это позволяет управляющим цепям процессора получать инструкции со скоростью самой медленной стадии обработки, однако при этом намного быстрее, чем при выполнении эксклюзивной полной обработки каждой инструкции от начала до конца.

Целью лабораторной работы является получение навыка организации асинхронной передачи данных между потоками на примере конвейера. Для ее достижения поставлены следующие задачи:

- выбрать логический алгоритм, выполняющий последовательные преобразования исходных данных;
- изучить строение конвейера;
- разработать схему реализации конвейерной системы;
- реализовать параллельный и последовательный конвейеры;
- провести тестирование разработанного программного обеспечения;
- провести анализ скорости работы реализаций конвейеров.

1 Аналитическая часть

Рассмотрим алгоритм нахождения обратной матрицы размером 3×3. Обратная матрица может быть найдена, если ее определитель не равен 0 (1.1). Определитель матрицы A размером 3×3 можно вычислить по формуле разложения по первой строке (1.2).

$$det \neq 0 \tag{1.1}$$

$$\begin{vmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{vmatrix} = A_{00} * \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} - A_{01} * \begin{vmatrix} A_{10} & A_{12} \\ A_{20} & A_{22} \end{vmatrix} + A_{02} * \begin{vmatrix} A_{10} & A_{11} \\ A_{20} & A_{21} \end{vmatrix} =$$

$$A_{00} * (A_{11} * A_{22} - A_{12} * A_{21}) - A_{01} * (A_{10} * A_{22} - A_{12} * A_{20}) + \\ + A_{02} * (A_{10} * A_{21} - A_{11} * A_{20}) \tag{1.2}$$

Общую формулу нахождения матрицы A^{-1}, обратной к матрице A, представим как произведение транспонированной матрицы алгебраических дополнений A_*^T на число, обратное определителю исходной матрицы [4] (1.3).

$$A^{-1} = \frac{1}{det} A_*^T \tag{1.3}$$

Разобьем алгоритм на 3 этапа преобразования исходной матрицы.

1.1 Нахождение матрицы алгебраических дополнений

Матрица алгебраических дополнений состоит из определителей матриц размером 2×2. Они получаются путем мысленного вычеркивания столбца и строки элемента, к которому в данный момент считается алгебраическое дополнение. Пример вычисления алгебраического дополнения A_{*00} (1.4), A_{*01} (1.5).

$$A_{*00} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \tag{1.4}$$

$$A_{*01} = -\begin{vmatrix} A_{10} & A_{12} \\ A_{20} & A_{22} \end{vmatrix} \tag{1.5}$$

При этом значение будет взято с обратным знаком в том случае, когда сумма индексов элемента нечетна.

1.2 Транспонирование матрицы

При транспонировании строки становятся соответствующими столбцами, а столбцы — строками. Так, при размере матрицы 3×3 элементы A_{01} и A_{10}, A_{02} и A_{20}, A_{12} и A_{21} меняются местами.

1.3 Умножение матрицы на число

При умножении матрицы на число каждый элемент матрицы умножается на это число.

1.4 Организация конвейерной системы

Конвейер будет состоять из 4 уровней. Обработанные данные передаются с одного уровня (ленты) конвейера на следующий. На каждой ленте осуществляется обработка данных. Для каждой ленты существует своя очередь необработанных заявок. После обработки на последней ленте конвейера заявки поступают в результирующую очередь. Уровни конвейера:

1. 0 уровень — генерация входных данных в первую очередь.
2. 1 уровень — преобразование исходной матрицы к матрице алгебраических дополнений.
3. 2 уровень — транспонирование матрицы алгебраических дополнений.
4. 3 уровень — умножение полученной матрицы на число, обратное определителю исходной матрицы.

1.5 Вывод по аналитической части

В данном разделе была рассмотрена последовательность действий при нахождении обратной матрицы размером 3×3, а также описана структура конвейерной системы, с помощью которой можно решить данную задачу.

2 Конструкторская часть

На основе полученных аналитических данных представим схему работы конвейера, опишем используемые типы и структуры данных, разработаем тесты для проверки корректности работы программы.

2.1 Схемы алгоритмов

На рисунках 2.1 – 2.3 представлены схемы алгоритмов работы конвейера и ленты.

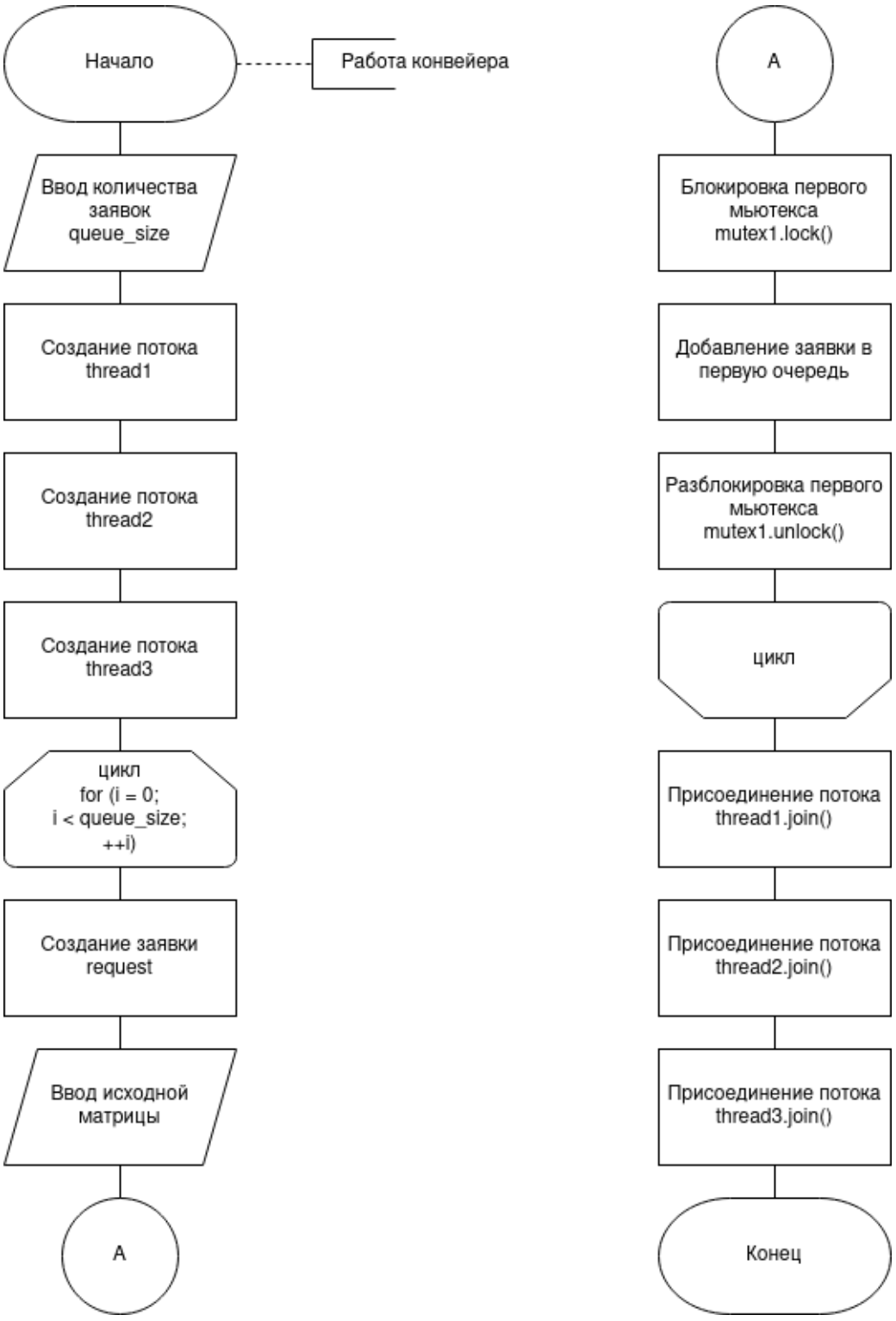

Рисунок 2.1 — Алгоритм работы конвейера

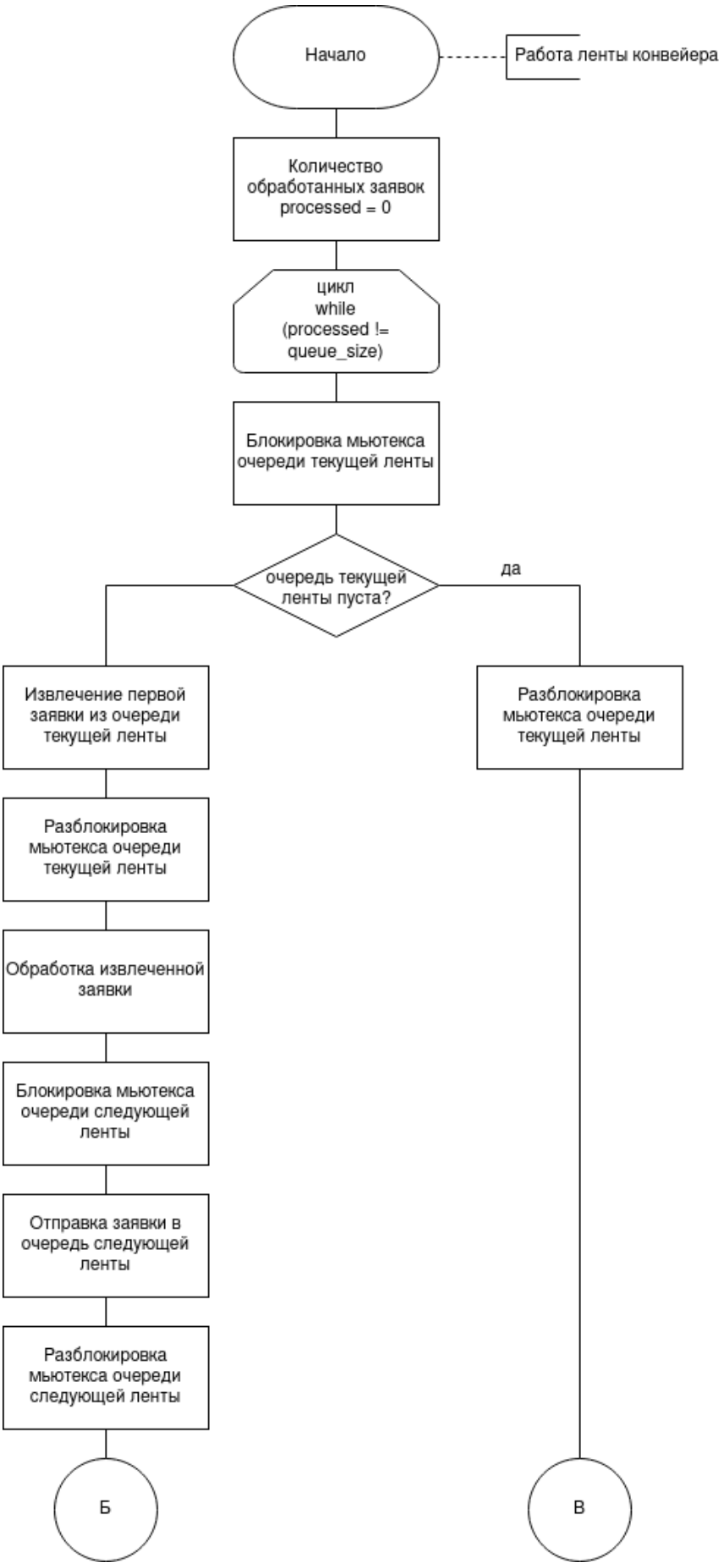

Рисунок 2.2 — Алгоритм работы ленты конвейера. Часть 1

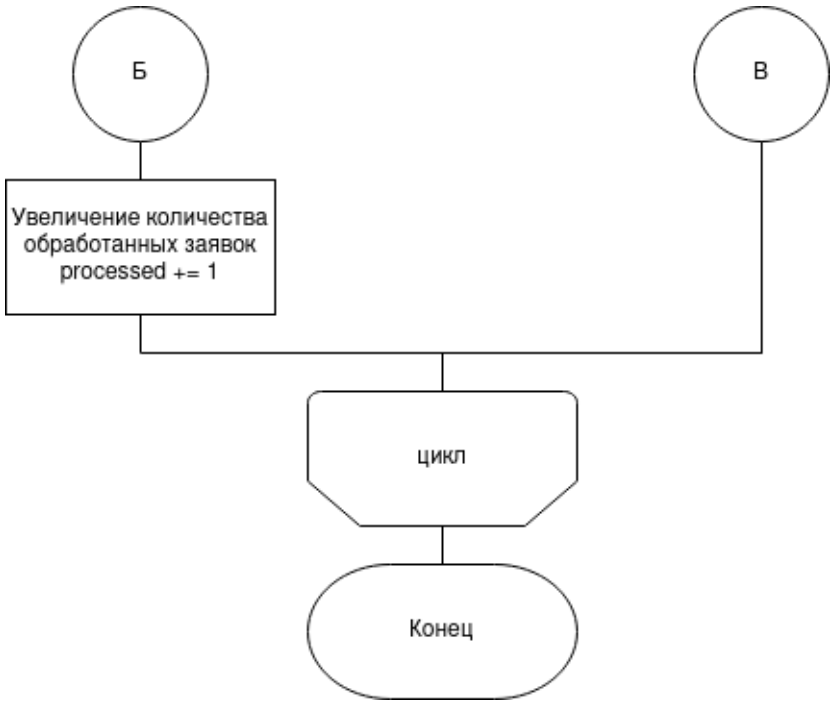

Рисунок 2.3 — Алгоритм работы ленты конвейера. Часть 2

2.2 Типы и структуры данных

Для реализации очередей линий конвейера используется встроенный тип данных `queue`. Этот тип данных позволяет работать с контейнером, как с очередью, при этом имеет достаточный интерфейс.

Для реализации мьютексов используется встроенный тип данных `mutex`, предоставляющий полный функционал для работы с мьютексами.

Для реализации потоков используется тип данных `thread`, позволяющий создавать потоки, которые могут параллельно выполнять определенные задачи.

Для реализации заявок используется тип данных `request`. Память под матрицу выделяется динамически. Матрица — это массив (размером 3) элементов типа указатель на массив (размером 3) типа `float`. Такое хранение матрицы позволяет легко, используя двойные квадратные скобки, получать доступ к любому элементу матрицы. Для заявки хранятся значения времени начала и конца обработки на каждой линии: `in_line[1..3]` — момент времени начала обработки заявки, `out_line[1..3]` — момент времени конца обработки заявки. Также класс должен содержать такие методы для обработки заявки, как деление матрицы на определитель, вычисление матрицы алгебраических дополнений, транспонирование матрицы.

2.3 Способ тестирования

Создаваемое программное обеспечение будет протестировано методом `черного ящика`. Для тестирования были выделены следующие классы эквивалентности:

1. Неверный выбор режима работы программы — пустой ввод, нецифровой символ или вещественное число.
2. Верный выбор режима работы программы — цифра из диапазона [1..5] или число, выходящее за пределы указанного диапазона.
3. Неверный ввод количества исходных матриц — пустой ввод, нецифровой символ, целое неположительное или вещественное число.
4. Верный ввод количества исходных матриц — целое положительное число.
5. Неверный ввод элемента исходной матрицы — пустой ввод, нецифровой символ или вещественное число.
6. Верный ввод элемента исходной матрицы — целое число.

2.4 Тестовые данные

В таблицах 2.1 – 2.3 представлены тестовые данные.

Таблица 2.1 — Функциональные тесты выбора режима работы программы

Номер тестового случая	Ввод	Ожидаемый результат
1	_[1]	Неверный ввод
2	α	Неверный ввод
3	3.14	Неверный ввод
4	1	Введите элементы матрицы
5	2	Введите элементы матрицы
6	3	Введите количество матриц
7	4	Введите количество матриц
8	5	Таблица замеров времени
9	0	.[2]

Таблица 2.2 — Функциональные тесты ввода количества исходных матриц

Номер тестового случая	Ввод	Ожидаемый результат
1	_[1]	Неверный ввод
2	α	Неверный ввод
3	0	Неверный ввод
4	3.14	Неверный ввод
5	2	Введите элементы матрицы

Таблица 2.3 — Функциональные тесты ввода элементов исходной матрицы

Номер тестового случая	Ввод	Ожидаемый результат
1	_[1]	Неверный ввод
2	α	Неверный ввод
3	2.2	Неверный ввод
4	0	...[3]

[1]Пустой ввод

[2]Конец работы программы

[3]Ожидание ввода следующего элемента

2.5 Структура программного обеспечения

Программное обеспечение разработано с использованием объектно-ориентированного подхода. Программа содержит функции, отвечающие за линии последовательного `line[1..3]Order` и параллельного `line[1..3]` конвейеров. Они ничего не принимают на вход и ничего не возвращают. Методы класса `request` — `setToCofactor`, `setToTranspose`, `divideByDet` — ничего не принимают на вход, а лишь изменяют матрицу той заявки, для которой этот метод был вызван.

2.6 Вывод по конструкторской части

В данном разделе были разработаны схемы алгоритмов работы конвейера и лент конвейера, а также подготовлены тестовые данные для программного обеспечения.

3 Технологическая часть

В данном разделе приведены средства реализации и листинги кода, а также перечислены требования к разрабатываемому программному обеспечению.

3.1 Требования к ПО

К программе предъявляется ряд требований:

- требования ко вводу
  - должен быть выбран режим работы программы (целое число в диапазоне [1..5]);
  - должны быть введены следующие значения: количество исходных матриц для преобразования при необходимости (целое положительное число), элементы матрицы (целые числа).
- требования к выводу
  - вычисленные обратные матрицы;
  - таблица времен работы реализаций конвейеров на случайных значениях в наносекундах.
- ограничения работы программы
  - при вводе режима работы программы, выходящего за пределы указанного диапазона, программа завершает работу.
- функциональные требования к программному обеспечению
  - при запуске программа должна выводить меню с возможными режимами работы;
  - программа должна находить одну или несколько обратных матриц с помощью последовательного или параллельного конвейера;
  - в исследовательском режиме программа должна замерять время обработки всех запросов последовательным и параллельным конвейером при случайных элементах матриц.

3.2 Средства реализации

Для реализации ПО был выбран язык программирования `C++` [1]. Это обусловлено наличием широкого спектра возможностей для работы с потоками и мьютексами, а такде поддержкой объектно-ориентированного подхода программирования.

В качестве среды разработки была выбрана `Visual Studio Code` [3]. Достаточный опыт работы в этой среде, удобства написания кода и его автодополнения стали ключевыми при выборе.

3.3 Реализация алгоритмов

В листингах 3.1 – 3.11 приведены реализации функций, использовавшихся в программном обеспечении.

Листинг 3.1 — Функция преобразования матрицы к матрице алгебраических дополнений

```
1  void request::setToCofactor() {
2    float temp1 = values[1][1]*values[2][2] -\
3    values[1][2]*values[2][1];
4    float temp2 = -(values[1][0]*values[2][2] -\
5    values[1][2]*values[2][0]);
6    float temp3 = values[1][0]*values[2][1] -\
7    values[1][1]*values[2][0];
8    float temp4 = -(values[0][1]*values[2][2] -\
9    values[0][2]*values[2][1]);
10   float temp5 = values[0][0]*values[2][2] -\
11   values[0][2]*values[2][0];
12   float temp6 = -(values[0][0]*values[2][1] -
13   values[0][1]*values[2][0]);
14   values[2][0] = values[0][1]*values[1][2] -\
15   values[0][2]*values[1][1];
16   values[2][1] = -(values[0][0]*values[1][2] -\
17   values[0][2]*values[1][0]);
18   values[2][2] = values[0][0]*values[1][1] -\
19   values[0][1]*values[1][0];
20   values[0][0] = temp1;
21   values[0][1] = temp2;
22   values[0][2] = temp3;
23   values[1][0] = temp4;
24   values[1][1] = temp5;
25   values[1][2] = temp6;
26 }
```

Листинг 3.2 — Функция транспонирования матрицы

```
1  void request::setToTranspose() {
2    float temp = values[0][1];
3    values[0][1] = values[1][0];
4    values[1][0] = temp;
5    temp = values[0][2];
6    values[0][2] = values[2][0];
7    values[2][0] = temp;
8    temp = values[1][2];
```

```
9         values[1][2] = values[2][1];
10        values[2][1] = temp;
11      }
```

Листинг 3.3 — Функция деления матрицы на определитель исходной матрицы

```
1     void request::divideByDet() {
2       for (int i = 0; i < 3; ++i) {
3         for (int j = 0; j < 3; ++j) {
4           if (fabs(det) < EPS)
5           values[i][j] = 0;
6           else
7           values[i][j] /= det;
8         }
9       }
10    }
```

Листинг 3.4 — Функция первой ленты параллельного конвейера

```
1     void line1() {
2       int processed = 0;
3       while (processed != queue_size) {
4         mutex1.lock();
5         if (queue1.empty()) {
6           mutex1.unlock();
7           continue;
8         }
9         request *temp = queue1.front();
10        queue1.pop();
11        mutex1.unlock();
12        temp->setToCofactor();
13        mutex2.lock();
14        queue2.push(temp);
15        mutex2.unlock();
16        processed++;
17      }
18    }
```

Листинг 3.5 — Функция второй ленты параллельного конвейера

```
1     void line2() {
2       int processed = 0;
3       while (processed != queue_size) {
4         mutex2.lock();
5         if (queue2.empty()) {
6           mutex2.unlock();
7           continue; }
8         request *temp = queue2.front();
9         queue2.pop();
10        mutex2.unlock();
11        temp->setToTranspose();
12        mutex3.lock();
13        queue3.push(temp);
14        mutex3.unlock();
15        processed++;
16      }
17    }
```

Листинг 3.6 — Функция третьей ленты параллельного конвейера

```
1     void line3() {
2       int processed = 0;
3       while (processed != queue_size) {
4         mutex3.lock();
5         if (queue3.empty()) {
6           mutex3.unlock();
7           continue;
8         }
9         request *temp = queue3.front();
10        queue3.pop();
11        mutex3.unlock();
12        temp->divideByDet();
13        queue4.push(temp);
14        processed++;
15      }
16    }
```

Листинг 3.7 — Функция первой ленты последовательного конвейера

```
1     void line1Order() {
2       int processed = 0;
3       while (processed != queue_size) {
4         request *temp = queue1.front();
5         queue1.pop();
6         temp->setToCofactor();
7         queue2.push(temp);
8         processed++;
9       }
10    }
```

Листинг 3.8 — Функция второй ленты последовательного конвейера

```
1     void line2Order() {
2       int processed = 0;
3       while (processed != queue_size) {
4         request *temp = queue2.front();
5         queue2.pop();
6         temp->setToTranspose();
7         queue3.push(temp);
8         processed++;
9       }
10    }
```

Листинг 3.9 — Функция третьей ленты последовательного конвейера

```
1     void line3Order() {
2       int processed = 0;
3       while (processed != queue_size) {
4         request *temp = queue3.front();
5         queue3.pop();
6         temp->divideByDet();
7         queue4.push(temp);
8         processed++;
9       }
10    }
```

Листинг 3.10 — Параллельный конвейер

```
    thread thread1(line1);
    thread thread2(line2);
    thread thread3(line3);
    request *temp;
    for (int i = 0; i < queue_size; ++i) {
      temp = new request;
      temp->input();
      mutex1.lock();
      queue1.push(temp);
      mutex1.unlock();
    }
    thread1.join();
    thread2.join();
    thread3.join();
```

Листинг 3.11 — Последовательный конвейер

```
    request *temp;
    for (int i = 0; i < queue_size; ++i) {
      temp = new request;
      temp->input();
      queue1.push(temp);
    }
    line1();
    line2();
    line3();
```

3.4 Пример работы программы

На рисунке (3.1) приведен пример работы программы.

```
Menu:
    1. Calculate inverse matrix by order.
    2. Calculate inverse matrix by threads.
    3. Calculate some inverse matrixes by order.
    4. Calculate some inverse matrixes by threads.
    5. Complex test both executions.

Your choice: 4
Input count matrixes: 2
Input matrix:
2 3 -6
8 -4 2
1 1 3
Input matrix:
-8 4 -1
1 2 5
6 -1 2

Calculated matrix:
0.0843373 0.0903614 0.108434
0.13253 -0.0722892 0.313253
-0.0722892 -0.0060241 0.192771

Calculated matrix:
0.169811 -0.132075 0.415094
0.528302 -0.188679 0.735849
-0.245283 0.301887 -0.377358

Menu:
    1. Calculate inverse matrix by order.
    2. Calculate inverse matrix by threads.
    3. Calculate some inverse matrixes by order.
    4. Calculate some inverse matrixes by threads.
    5. Complex test both executions.

Your choice: 0
Для закрытия данного окна нажмите <ВВОД>...
```

Рисунок 3.1 — Пример работы программы

3.5 Вывод по технологической части

В данном разделе были описаны средства реализации, представлены требования к ПО и реализованы последовательный и параллельный конвейер для нахождения обратной матрицы.

4 Исследовательская часть

В этом разделе будет исследовано быстродействие разработанных реализаций конвейеров.

4.1 Технические характеристики

Ниже приведены технические характеристики устройства, на котором было проведено тестирование ПО:

- операционная система Windows 10 64-разрядная;
- оперативная память 16 ГБ;
- процессор Intel(R) Core(TM) i5-4690 @ 3.50ГГц.

4.2 Время выполнения реализаций конвейеров

Чтобы точно оценить время выполнения реализаций двух конвейеров была использована функция `std::chrono::high_resolution_clock::now()` [2], которая возвращает точку, указывающую на текущее время. Так, с помощью двух последовательных вызовов можно определить, в период какого времени выполнялась функция.

Замеры проводились при длине очереди в 10 заявок.

В таблицах 4.1 и 4.2 приведены результаты замеров. Время в таблицах указано от начала работы параллельного и последовательного конвейера соответственно.

Таблица 4.1 — Работа параллельного конвейера (в наносекундах)

Линия №	Заявка №	Начало обработки	Конец обработки
1	1	146430	147208
2	1	156072	156660
3	1	166374	167627
1	2	164384	164979
2	2	170763	171335
3	2	175956	176864
1	3	169114	169666
2	3	177881	178291
3	3	182391	183223
1	4	170483	171173
2	4	183906	184471
3	4	188749	189563
1	5	172411	172992
2	5	190423	190833
3	5	199256	199825
1	6	173604	174066
2	6	192104	192340
3	6	203401	203771
1	7	174556	175108
2	7	198124	198353
3	7	204599	204941
1	8	175881	176433
2	8	198864	199259
3	8	205389	205825
1	9	177228	177764
2	9	199798	200200
3	9	206364	206916
1	10	178595	179161
2	10	205833	206239
3	10	207631	208114

Таблица 4.2 — Работа последовательного конвейера (в наносекундах)

Линия №	Заявка №	Начало обработки	Конец обработки
1	1	2125	2708
2	1	10625	10832
3	1	15170	15843
1	2	3144	3712
2	2	11145	11343
3	2	16088	16473
1	3	4019	4603
2	3	11588	11777
3	3	16712	17005
1	4	4902	5378
2	4	12027	12218
3	4	17242	17527
1	5	5685	6103
2	5	12457	12645
3	5	17753	18037
1	6	6445	6856
2	6	12875	13069
3	6	18277	18559
1	7	7159	7583
2	7	13304	13492
3	7	18802	19077
1	8	7860	8243
2	8	13721	13912
3	8	19315	19596
1	9	8527	8934
2	9	14147	14337
3	9	19827	20106
1	10	9200	9554
2	10	14582	14755
3	10	20343	20626

При этом суммарное время работы параллельного конвейера составило 282084 наносекунды, а последовательного — 20720 наносекунд.

Для чистоты эксперимента был запущен тест при длине очереди в 50 заявок. Вместо обрабатывающих функций была использована команда `Sleep(100)`, которая имитирует выполнение задачи. При этом были получе-

ны следующие результаты: время работы параллельного конвейера составило 116150200 наносекунд, последовательного — 326595400 наносекунд, что примерно в 3 раза дольше.

4.3 Вывод по исследовательской части

По результатам замеров можно сказать, что сложность задачи, решаемой каждой лентой конвейера, недостаточна для того, чтобы перекрыть расходы на создание потоков, блокировку и разблокировку мьютексов. При большом количестве заявок в очереди, а также достаточно сложной задаче, параллельный конвейер показывает результат лучше, чем последовательный. При несложной задаче и малом количестве заявок последовательный конвейер покажет лучший результат, поскольку в этом случае не нужно создавать потоки, работать с мьютексами.

Заключение

В ходе проделанной работы была достигнута поставленная цель и решены следующие задачи:

- изучено строение конвейера;
- разработана схема реализации конвейерной системы;
- реализованы последовательный и параллельный конвейеры;
- проведено тестирование разработанного программного обеспечения;
- проведен анализ скорости работы реализаций конвейеров.

Было доказано, что выделять потоки следует только в том случае, если каждому потоку будет дана сложная задача, в противном случае намного больше времени будет затрачиваться на создание потоков. При объемной задаче параллельный конвейер показал результат лучше, чем последовательный, что доказывает целесообразность использования потоков для более быстрого решения больших задач.

Список литературы

[1] C++ [Электронный ресурс]. Режим доступа: https://isocpp.org/. Дата обращения: 21.12.2021.

[2] Функция now() [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/cpp/chrono/high-resolution-clock/now. Дата обращения: 21.12.2021.

[3] Visual Studio Code - Code Editing [Электронный ресурс]. Режим доступа: https://code.visualstudio.com. Дата обращения: 21.12.2021.

[4] Обратная матрица [Электронный ресурс]. Режим доступа: http://www.mathprofi.ru/kak-naiti-obratnuyu-matricu.html. Дата обращения: 21.12.2021.