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 Артюхин Н.П.

Группа ИУ7-51Б

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

Москва — 2022 г.

Оглавление

Введение

Целью данной работы является получение навыка организации асинхронного взаимодействия между потоками на примере моделирования конвейера.

Конвейры широко применяются программистами для решения трудоемких задач, которые можно разделить на этапы, а также в большинстве современных быстродействующих процессоров.

Для достижения поставленной цели требуется решить следующие задачи:

1) изучение основ конвейерной обработки данных;

2) изучить алгоритмы возведения матрицы в квадрат, приведения матрицы к верхнетреугольному виду и нахождения определителя матрицы;

3) разработка параллельной и последовательной версий конвейера с тремя стадиями обработки;

4) реализация параллельной и последовательной версий конвейера с тремя стадиями обработки;

5) сравнительный анализ времени работы последовательной и параллельной реализаций конвейера при различном количестве задач;

6) собрать статистику времени ожидания заявок (максимальное, минимальное, среднее, медианное значение) в каждой из очередей (лент) конвейера по отдельности, во всех очередях и времени обработки в системе.

1 Аналитическая часть

В данном разделе будет представлено описание конвейерной обработки данных и этапов обработки матрицы.

1.1 Конвейерная обработка данных

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. Так, обработку любой машинной команды можно разделить на несколько этапов, организовав передачу данных от одного этапа к следующему [1].

Конвейерную обработку можно использовать для совмещения этапов выполнения разных команд. Производительность при этом возрастает, так как одновременно на различных лентах конвейера выполняются несколько команд. Конвейеризация увеличивает пропускную способность процессора, однако она не сокращает время выполнения отдельной команды.

1.2 Описание этапов обработки

В качестве примера для организации конвейрной обработки будет обрабатываться матрица.

Задача обработки матрицы будет разбита на 3 этапа (ленты):

1) возведение матрицы в квадрат;

2) приведение матрицы к верхнетреугольному виду;

3) нахождение определителя матрицы.

Определитель матрицы будет находиться методом Гаусса.

Чтобы вычислить определитель по методу Гаусса, исходная матрица приводится к верхнетреугольной форме с помощью элементарных преобразований, определитель исходной матрицы не изменится и будет равен

произведению элементов, расположенных на главной диагонали верхнетреугольной матрицы [2].

Матрица называется верхнетреугольной, если $a_{ij} > 0$ для всех $i > j$, где a_{ij} — элемент матрицы, расположенный в i-ой строке в j-ом столбце. На рисунке 1.1 представлена верхнетреугольная матрица.

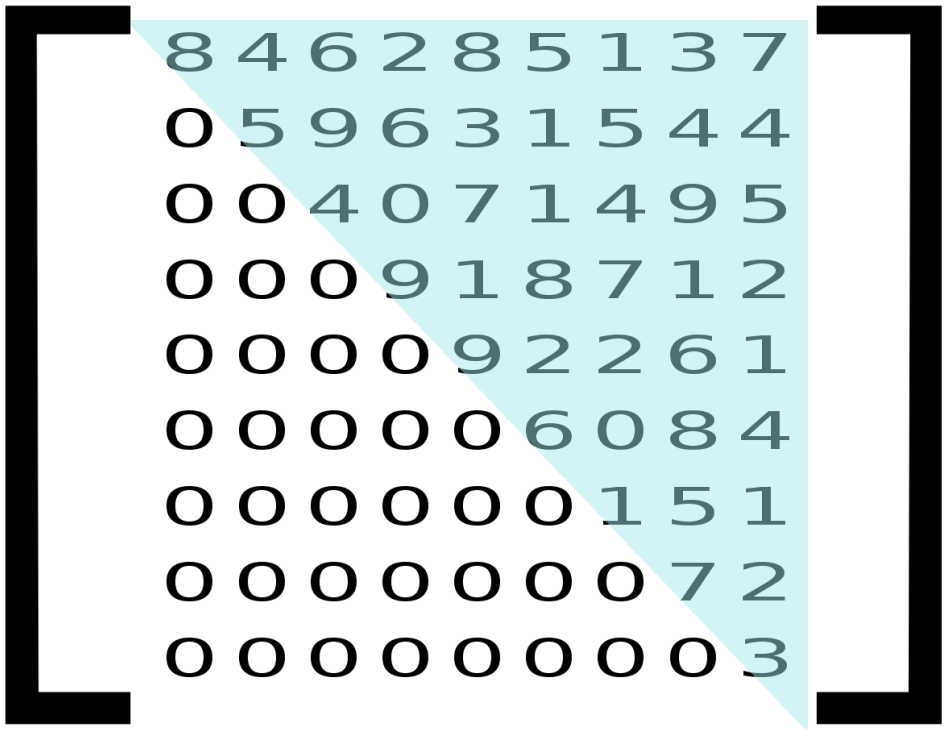

Рисунок 1.1 – Верхнетреугольная матрица

Вывод

В данном разделе было представлено описание конвейерной обработки данных и этапов обработки матрицы.

2 Конструкторская часть

В данном разделе будут представлены схемы алгоритмов последовательной и конвейерной обработки матриц.

2.1 Алгоритмы этапов обработки матриц

На рисунке 2.1 приведена схема алгоритма возведения матрицы в квадрат, на рисунке 2.2 — схема алгоритма приведения матрицы к верхнетреугольному виду и на рисунке 2.3 — схема алгоритма нахождения определителя верхнетеугольной матрицы.

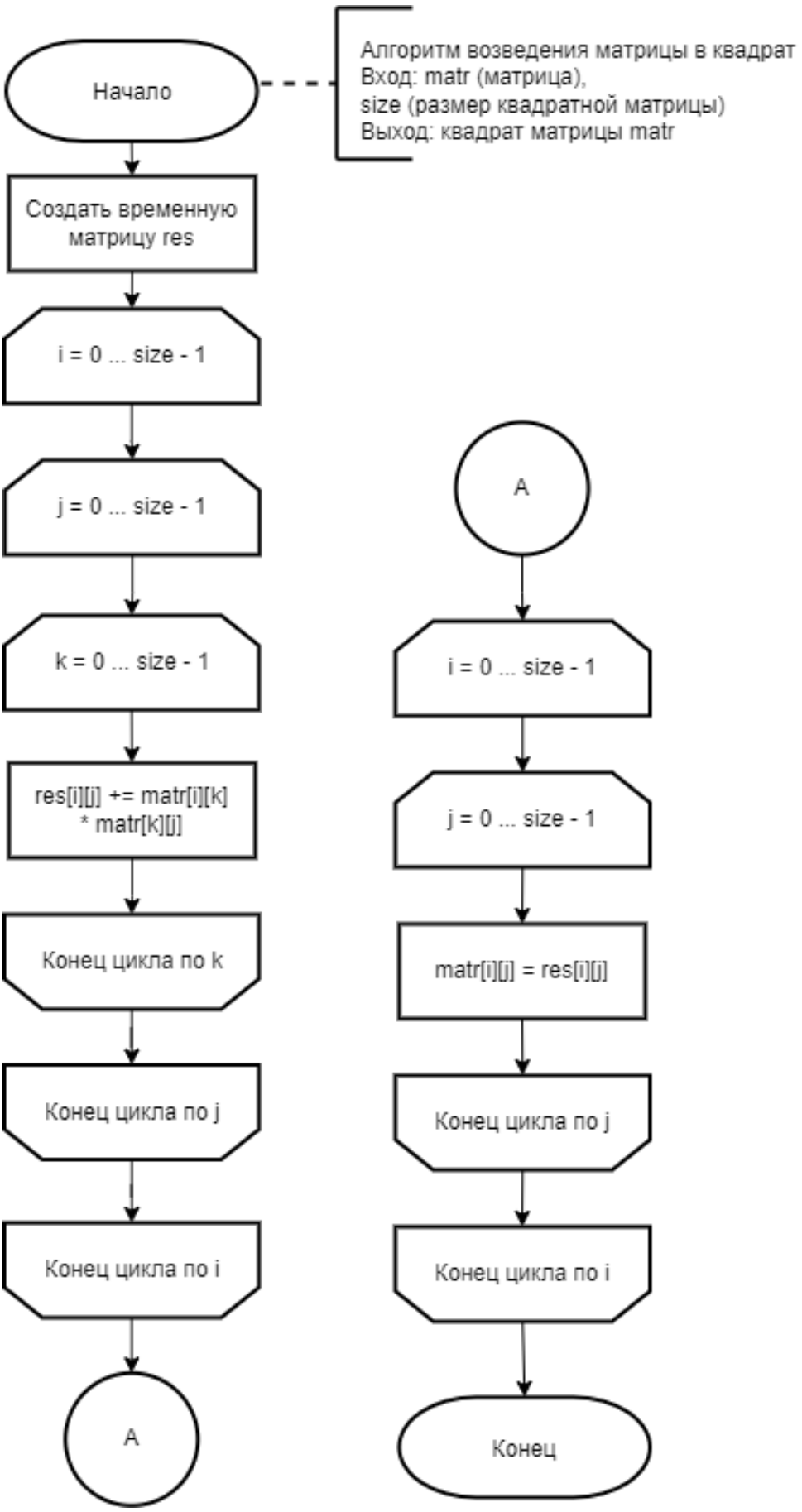

Рисунок 2.1 – Схема алгоритма возведения матрицы в квадрат

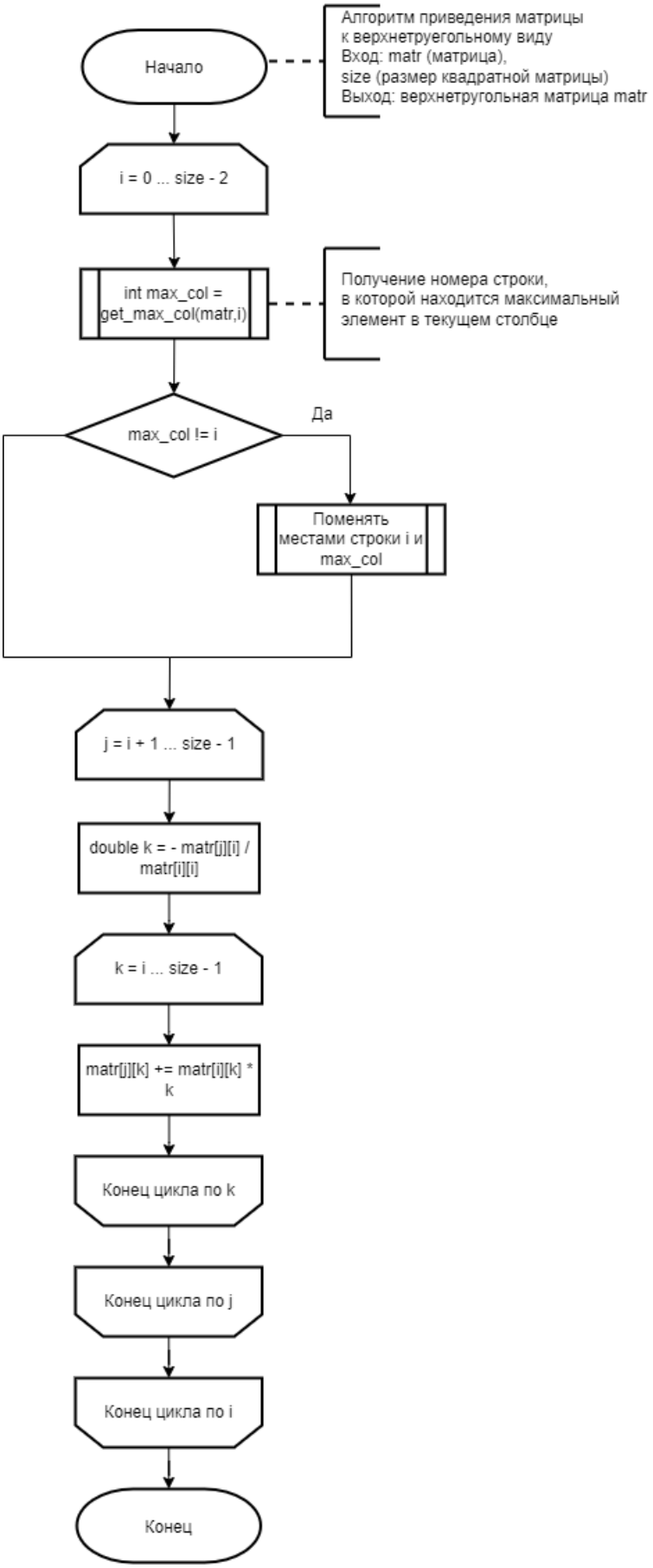

Рисунок 2.2 – Схема алгоритма приведения матрицы к верхнетреугольному виду

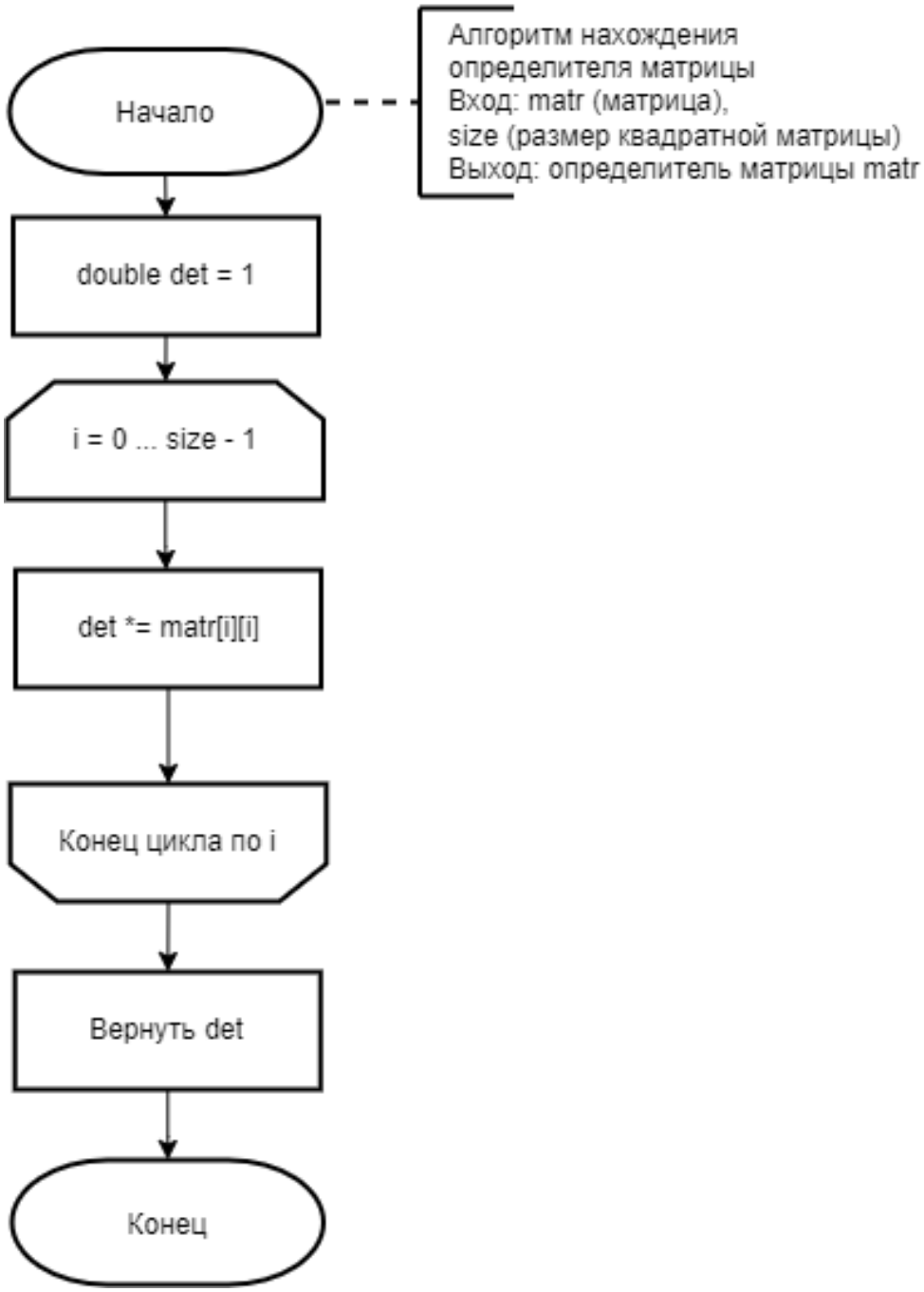

Рисунок 2.3 – Схема алгоритма нахождения определителя верхнетеугольной матрицы

2.2 Алгоритм последовательной обработки матриц

На рисунке 2.4 приведена схема алгоритма последовательной обработки матриц.

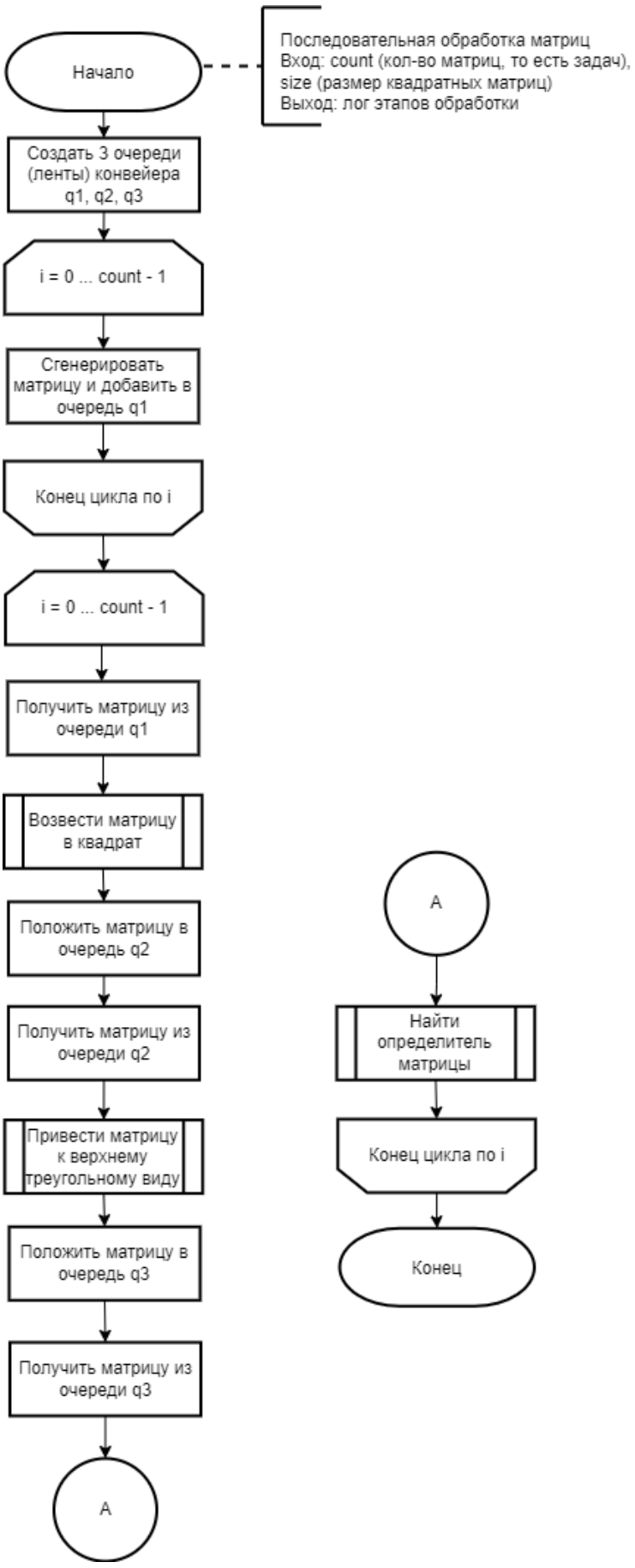

Рисунок 2.4 – Схема алгоритма последовательной обработки матриц

2.3 Алгоритм конвейерной обработки матриц

На рисунках 2.5 – 2.8 приведена схема алгоритма конвейерной обработки матриц.

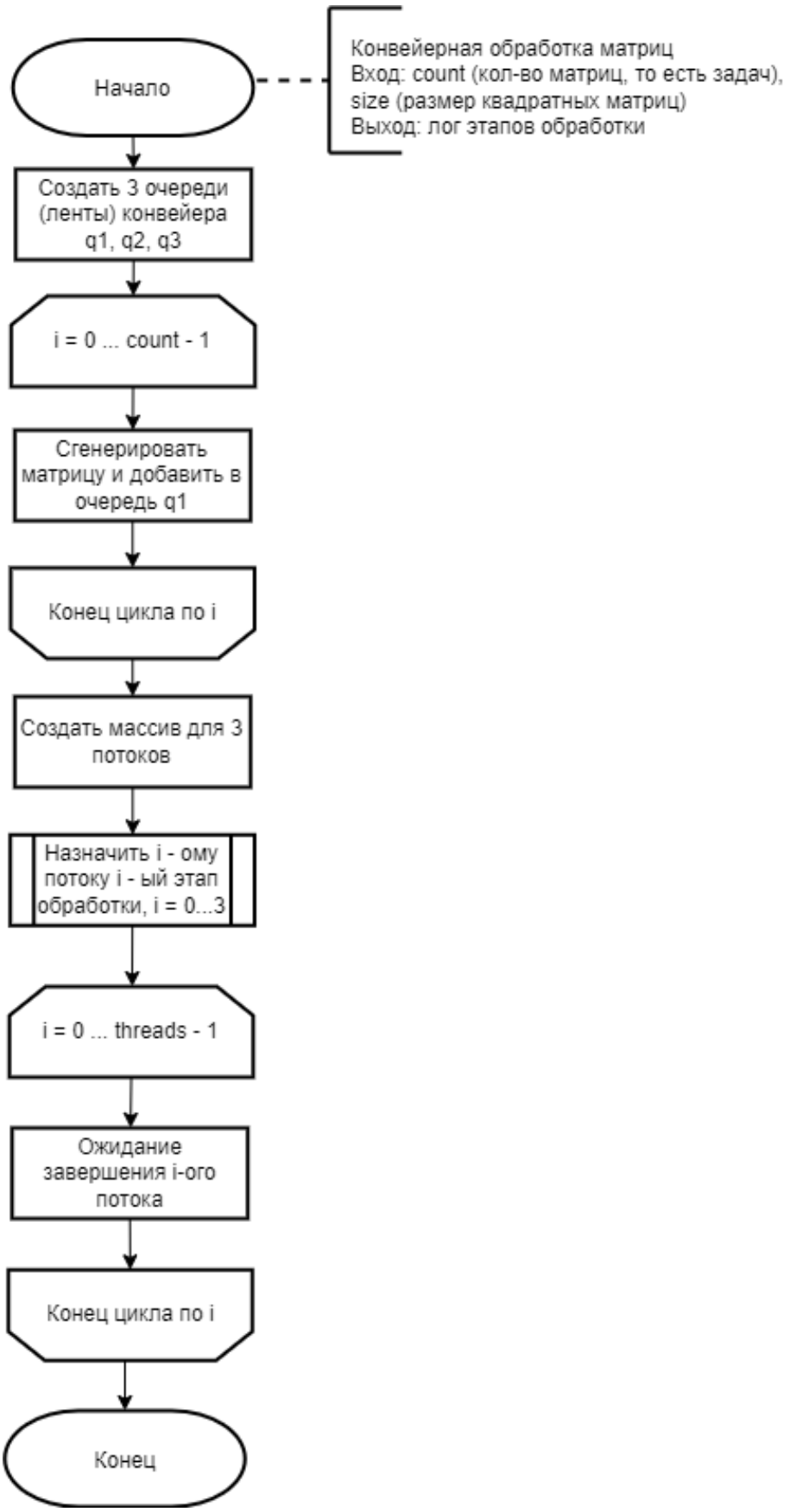

Рисунок 2.5 – Схема алгоритма конвейерной обработки матриц

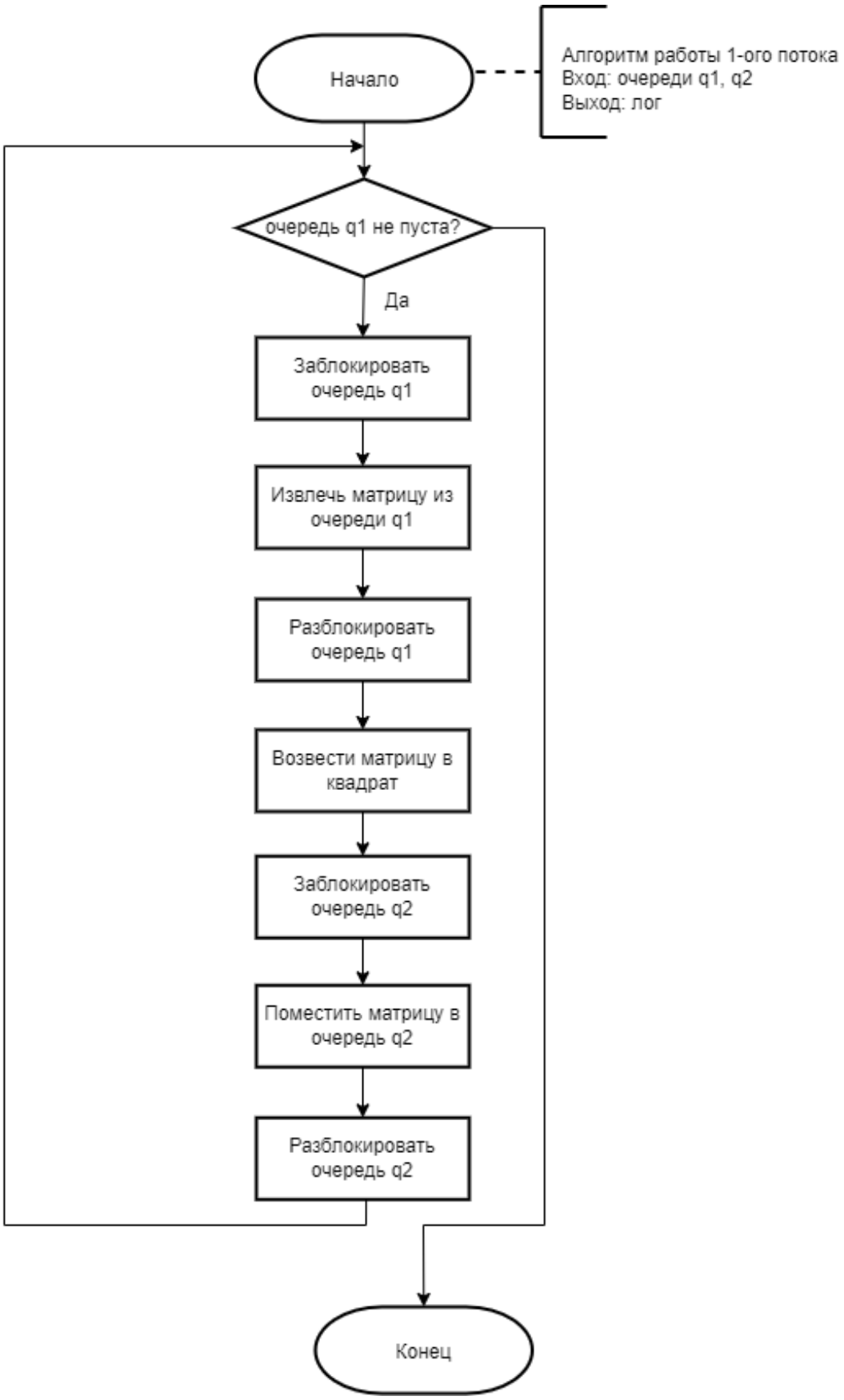

Рисунок 2.6 – Схема алгоритма конвейерной обработки матриц (1-ый поток)

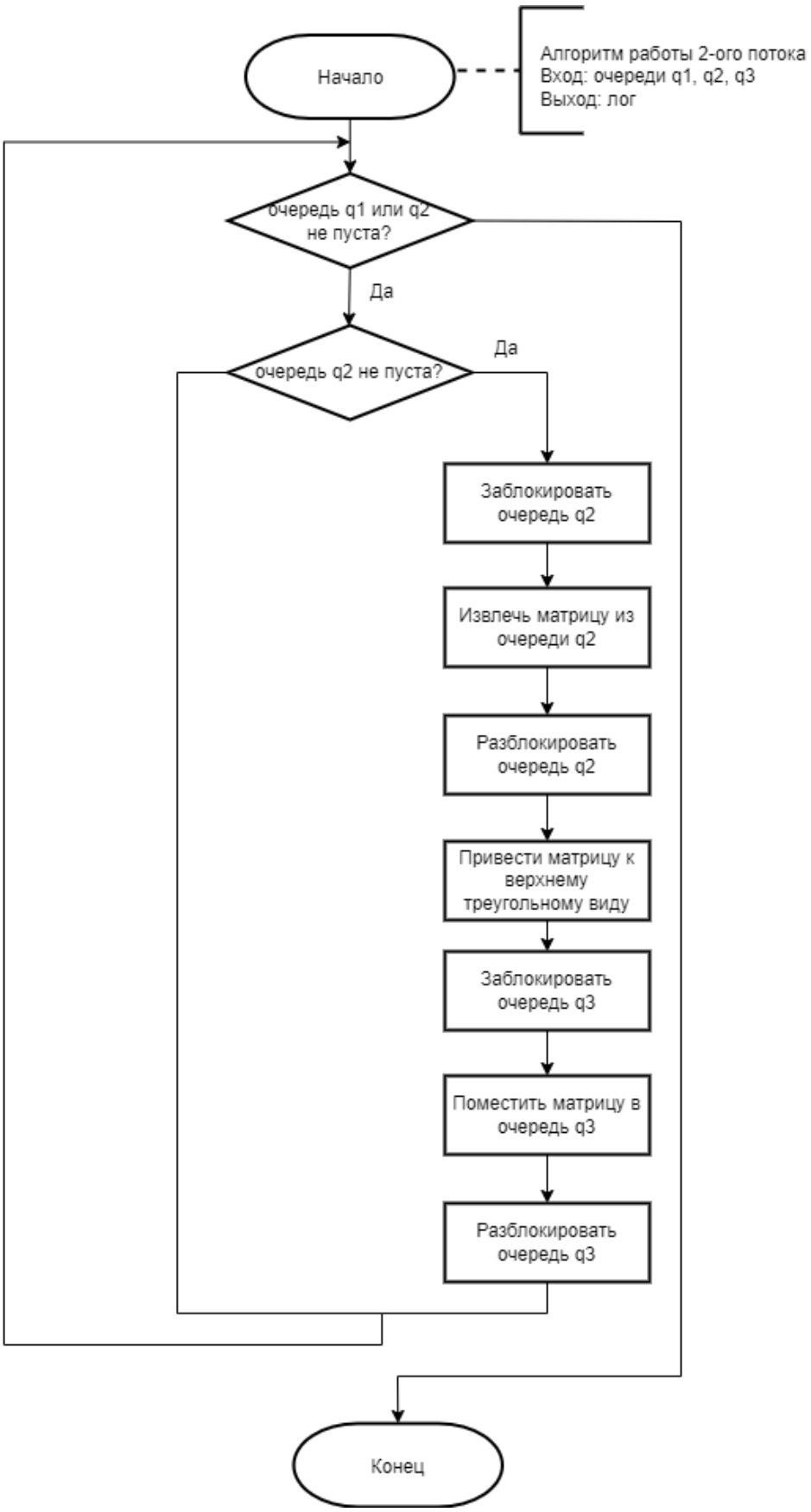

Рисунок 2.7 – Схема алгоритма конвейерной обработки матриц (2-ой поток)

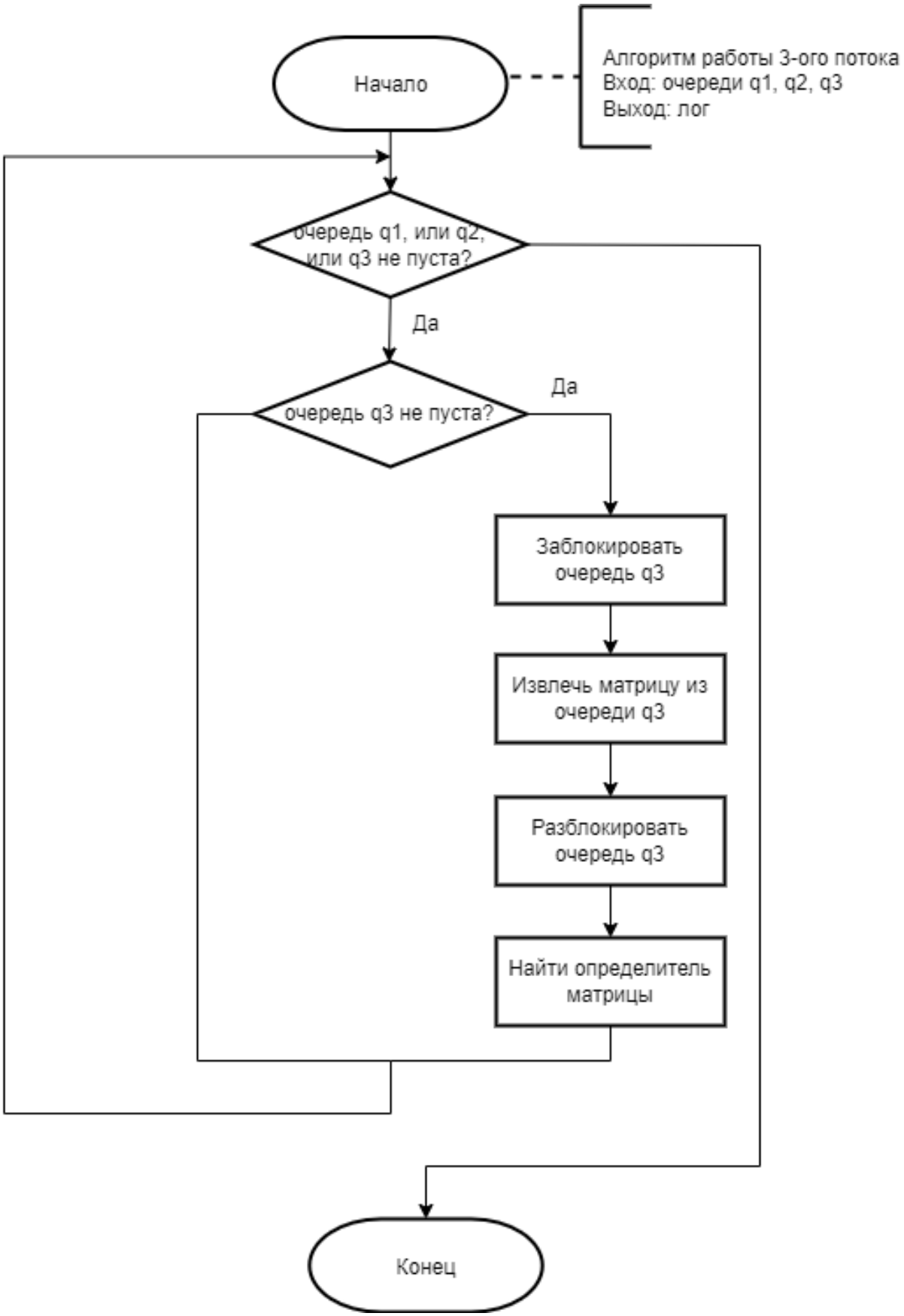

Рисунок 2.8 – Схема алгоритма конвейерной обработки матриц (3-ий поток)

Вывод

В данном разделе были разработаны схемы последовательного и конвейерного алгоритмов обработки матриц.

3 Технологическая часть

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

3.1 Требования к программному обеспечению

Вход: действие (0 — выход из программы, 1 — последовательная обработка матриц, 2 — конвейерная обработка матриц, 3 — сравнение последоватнельной и конвейерной обработки по времени работы, 4 — вывод информации об этапах обработки матриц), линейный размер квадратных матриц (в случае замеров времени – начальный и конечный размер) — целое положительное число, количество матриц (в случае замеров времени – минимальное и максимальное количество) — целое положительное число.

Выход: лог этапов обработки матриц, в случае выбора конвейерной обработки выводятся статистические данные — минимальное, максимальное, среднее, медианное значение времени нахождения заявки в каждой из очередей, во всех очередях, в системе.

3.2 Выбор средств реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык программирования C++ [3]. Данный язык программирования позволяет замерить время выполнения работы реализации алгоритма, а также организовать конвейерную обработку данных.

Время выполнения реализаций трехэтапного алгоритма обработки матриц было замерено с помощью функции *std::chrono::system_clock::now(...)* из библиотеки *chrono* [4].

В качестве среды разработки был выбран CLion [5].

Графики были построены с использованием языка программирования Python [6].

3.3 Реализация алгоритмов

В листингах 3.1 – 3.3 представлена реализация алгоритмов этапов обработки матриц, в листинге 3.4 — реализация алгоритма последовательной обработки матриц, а в листингах 3.5 – 3.8 — реализация алгоритма конвейерной обработки матриц.

Листинг 3.1 – Реализация алгоритма возведения матрицы в квадрат

```
void mult_matrix(matrix_t &matrix)
{
    std::vector<std::vector<double>> tmp_data;
    tmp_data.resize(matrix.size);
    for (size_t i = 0; i < matrix.size; i++)
    {
        tmp_data[i].resize(matrix.size);
    }
    matrix_t res;
    res.size = matrix.size;
    res.data = tmp_data;
    for (size_t i = 0; i < res.size; i++)
    {
        for (size_t j = 0; j < res.size; j++)
        {
            for (size_t k = 0; k < res.size; k++)
            {
                res.data[i][j] += matrix.data[i][k] * matrix.data[
                    k][j];
            }
        }
    }
    for (size_t i = 0; i < res.size; i++){
        for(size_t j = 0; j < res.size; j++){
            matrix.data[i][j] = res.data[i][j];
        }
    }
}
```

Листинг 3.2 – Реализация алгоритма приведения матрицы к верхнетреугольному виду

```
void triangulate_matr(matrix_t &matrix)
{
    for (size_t i = 0; i < matrix.size - 1; i++)
    {
        size_t max_col_place = get_max_in_column(matrix, i);

        if (max_col_place != i)
        {
            std::swap(matrix.data[i], matrix.data[max_col_place]);
        }

        for (size_t j = i + 1; j < matrix.size; j++)
        {
            double mult = -matrix.data[j][i] / matrix.data[i][i];

            for (size_t k = i; k < matrix.size; k++)
            {
                matrix.data[j][k] += matrix.data[i][k] * mult;
            }
        }
    }
}

int get_max_in_column(matrix_t matrix, int col_place)
{
    double max_elem = matrix.data[col_place][col_place];
    size_t max_place = col_place;

    for (size_t i = col_place + 1; i < matrix.size; i++)
    {
        if (matrix.data[i][col_place] >= max_elem)
        {
            max_elem = matrix.data[i][col_place];
            max_place = i;
        }
    }

    return max_place;
}
```

Листинг 3.3 – Реализация алгоритма нахождения определителя верхнетругольной матрицы

```
void get_determinate(matrix_t &matrix)
{
    double det = 1;

    for (size_t i = 0; i < matrix.size; i++)
    {
        det *= matrix.data[i][i];
    }

    matrix.det = det;
}
```

Листинг 3.4 – Реализация алгоритма последовательной обработки матриц

```
void linear_alg(int count, size_t size)
{
    std::queue<matrix_t> q1;
    std::queue<matrix_t> q2;
    std::queue<matrix_t> q3;

    queues_t queues = {.q1 = q1, .q2 = q2, .q3 = q3};

    for (int i = 0; i < count; i++)
    {
        matrix_t res = generate_matrix(size);

        queues.q1.push(res);
    }

    for (int i = 0; i < count; i++)
    {
        matrix_t matrix = queues.q1.front();
        stage1_linear(matrix, i + 1); // Stage 1
        queues.q1.pop();
        queues.q2.push(matrix);

        matrix = queues.q2.front();
        stage2_linear(matrix, i + 1); // Stage 2
        queues.q2.pop();
        queues.q3.push(matrix);
```

```
27
28          matrix = queues.q3.front();
29          stage3_linear(matrix, i + 1); // Stage 3
30          queues.q3.pop();
31      }
32  }
```

Листинг 3.5 – Реализация алгоритма конвейерной обработки матриц

```
1  void conveyor_alg(int count, size_t size)
2  {
3      std::queue<matrix_t> q1;
4      std::queue<matrix_t> q2;
5      std::queue<matrix_t> q3;
6
7      queues_t queues = {.q1 = q1, .q2 = q2, .q3 = q3};
8
9      for (int i = 0; i < count; i++)
10     {
11         matrix_t res = gen_matrix(size);
12         q1.push(res);
13     }
14
15     std::thread threads[THREADS];
16
17     threads[0] = std::thread(stage1_conv, std::ref(q1), std::ref(
           q2), std::ref(q3));
18     threads[1] = std::thread(stage2_conv, std::ref(q1), std::ref(
           q2), std::ref(q3));
19     threads[2] = std::thread(stage3_conv, std::ref(q1), std::ref(
           q2), std::ref(q3));
20
21     for (int i = 0; i < THREADS; i++)
22     {
23         threads[i].join();
24     }
25 }
```

Листинг 3.6 – Реализация алгоритма конвейерной обработки матриц (1-ый поток)

```
1  void stage1_conv(std::queue<matrix_t> &q1, std::queue<matrix_t> &
       q2)
```

```
{
    int task_num = 1;
    std::mutex m;

    while(!q1.empty())
    {
        m.lock();
        matrix_t matrix = q1.front();
        m.unlock();

        log_conveyor(matrix, task_num++, 1, mult_matrix);
        m.lock();
        q2.push(matrix);
        q1.pop();
        m.unlock();
    }
}
```

Листинг 3.7 – Реализация алгоритма конвейерной обработки матриц (2-ой поток)

```
void stage2_conv(std::queue<matrix_t> &q1, std::queue<matrix_t> &
   q2, std::queue<matrix_t> &q3)
{
    int task_num = 1;
    std::mutex m;

    do
    {
        m.lock();
        bool is_q2empty = q2.empty();
        m.unlock();

        if (!is_q2empty)
        {
            m.lock();
            matrix_t matrix = q2.front();
            m.unlock();

            log_conveyor(matrix, task_num++, 2, triangulate_matr);
            m.lock();
            q3.push(matrix);
```

```
21                 q2.pop();
22                 m.unlock();
23             }
24         } while (!q1.empty() || !q2.empty());
25 }
```

Листинг 3.8 – Реализация алгоритма конвейерной обработки матриц (3-ий поток)

```
1  void stage3_conv(std::queue<matrix_t> &q1, std::queue<matrix_t> &
       q2, std::queue<matrix_t> &q3)
2  {
3      int task_num = 1;
4      std::mutex m;
5
6      do
7      {
8          m.lock();
9          bool is_q3empty = q3.empty();
10         m.unlock();
11
12         if (!is_q3empty)
13         {
14             m.lock();
15             matrix_t matrix = q3.front();
16             m.unlock();
17
18             log_conveyor(matrix, task_num++, 3, get_determinate);
19             m.lock();
20             q3.pop();
21             m.unlock();
22         }
23     } while (!q1.empty() || !q2.empty() || !q3.empty());
24 }
```

3.4 Тестирование

В таблице 3.1 приведены функциональные тесты для алгоритмов последовательной и конвейерной обработки матриц. Все тесты были пройде-

ны успешно.

Таблица 3.1 – Тесты

№	Действие	Кол-во матриц	Размер матриц	Ожид. результат
1	1	5	-2	Некорректный ввод!
2	1	-1	5	Некорректный ввод!
3	-1	2	2	Неверное действие!
4	5	2	2	Неверное действие!
5	1	10	100	Лог этапов обработки
6	2	10	100	Лог этапов обработки
7	3	20 – 50	2	Замеры времени
8	3	10	10 – 100	Замеры времени
9	4			Этапы обработки (вывод)
10	0			Выход

Вывод

В данном разделе были представлены требования к программному обеспечению и средства реализации, реализованы и протестированы алгоритмы почледовательной и конвейерной обработки матриц.

4 Исследовательская часть

В текущем разделе будут представлены примеры работы разработанного программного обеспечения, постановка эксперимента и сравнительный анализ реализованных алгоритмов.

4.1 Пример работы программного обеспечения

На рисунке 4.1 представлен результат работы программы при последовательной обработке матриц, а на рисунке 4.2 — при конвейерной обработке матриц; линейный размер квадратных матриц — 100, количество матриц — 10. На данных рисунках первая колонка таблицы — номер заявки (матрицы), вторая — номер этапа (номер ленты конвейера), третья — время начала обработки заявки, четвертая — время завершения обработки заявки. Видно, что при последовательной обработке заявки обрабатываются последовательно, то есть сначала первая заявка проходит все этапы обработки, затем вторая и т.д. Также видно, что при конвейерной обработке заявки могут обрабатываться параллельно (на разных лентах конвейера). На рисунке 4.3 представлена статистика по времени ожидания заявок (минимальное, максимальное, среднее, медианное значения в секундах) в каждой из 3 очередей по отдельности, во всех очередях и по времени обработки в системе при конвейерной обработке матриц.

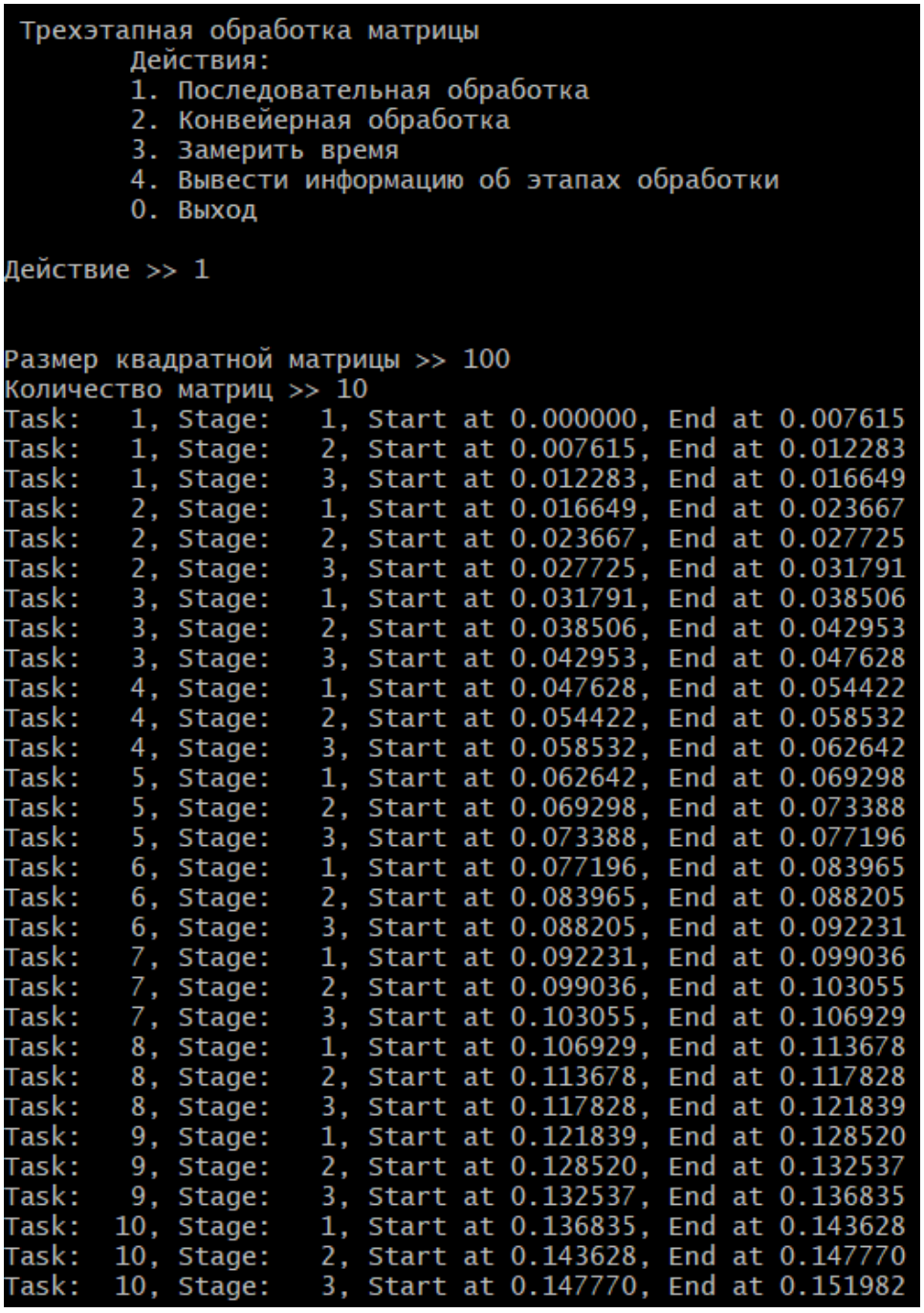

Рисунок 4.1 – Пример работы программы (последовательная обработка)

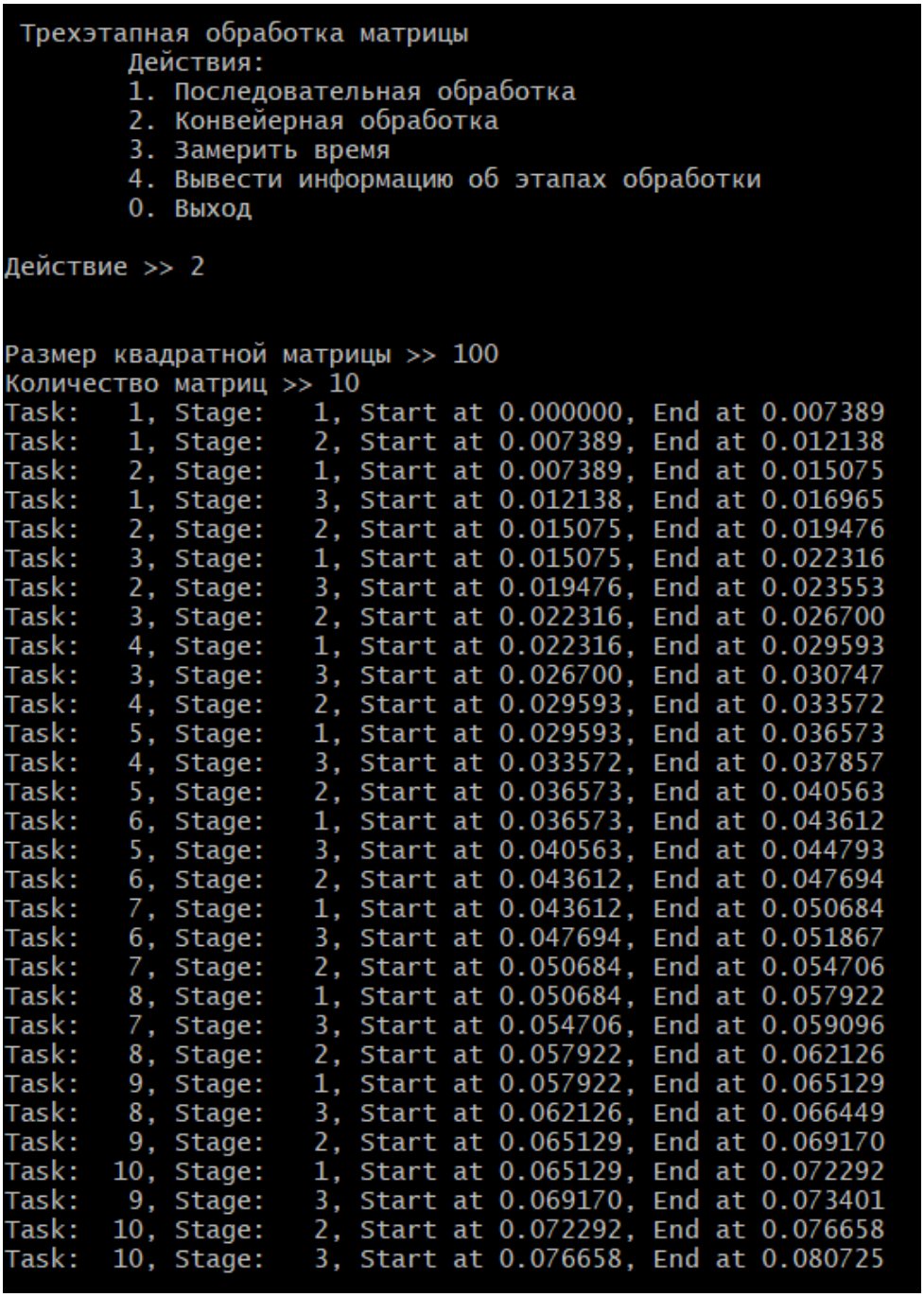

Рисунок 4.2 – Пример работы программы (конвейерная обработка)

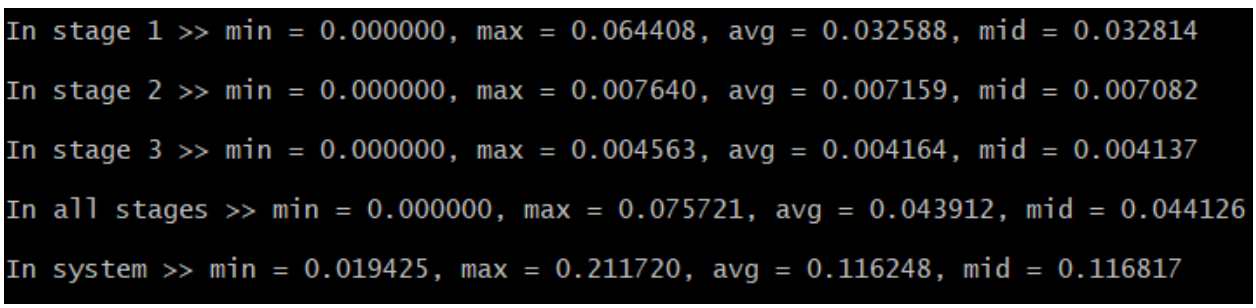

Рисунок 4.3 – Пример работы программы (статистика по очередям)

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнись замеры времени:

- операционная система — Windows 10 [7];
- оперативная память — 16 Гб;
- процессор — Intel® Core™ i5 10300H 2.5 ГГц;
- 4 физических ядра, 4 логических ядра.

Во время замеров времени выполнения реализаций алгоритмов ноутбук был включен в сеть питания и нагружен только встроенными приложениями окружения и системой тестирования.

4.3 Время выполнения реализаций алгоритмов построения спектра отрезков

Для замера времени выполнения реализованных алгоритмов использовалась функция *system_clock::now(...)* из библиотеки *chrono*. Данная функция возвращает время в наносекундах.

Использовать эту функцию необходимо дважды, затем из конечного времени нужно вычесть начальное, чтобы получить результат.

Замеры времени выполнения конвейерного и последовательного алгоритмов обработки матриц проводились для разного количества матриц (от 10 до 50 с шагом 5, размер матриц — 100 * 100). Время переводилось из наносекунд в секунды посредством деления на 10^9.

Результаты замеров времени приведены в таблице 4.1 (время в секундах).

Таблица 4.1 – Результаты замеров времени в секундах (количество матриц от 10 до 50 с шагом 5, размер — 100 * 100)

Количество матриц	Последовательный	Конвейерный
10	0.1489	0.0852
15	0.2205	0.1161
20	0.2945	0.1526
25	0.3676	0.2007
30	0.4384	0.2288
35	0.5104	0.2724
40	0.5825	0.2990
45	0.6599	0.3355
50	0.7327	0.3948

На рисунке 4.4 представлено сравнение времени работы последовательного и конвейрного алгоритмов обработки матриц.

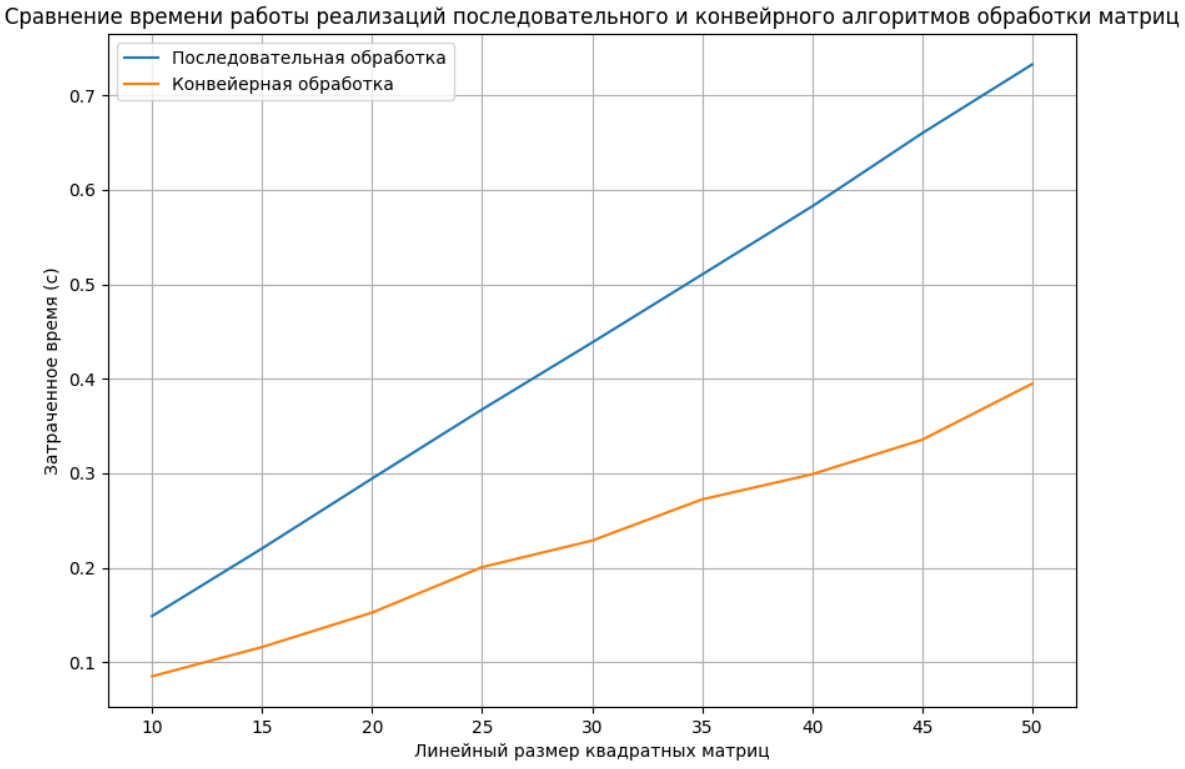

Рисунок 4.4 – Сравнение времени работы последовательного и конвейрного алгоритмов обработки матриц

На рисунке 4.5 представлена статистика по времени ожидания заявок (минимальное, максимальное, среднее, медианное значения в секундах) в

каждой из 3 очередей по отдельности, во всех очередях и по времени обработки в системе при конвейерной обработке матриц.

```
In stage 1 >> min = 0.000000, max = 0.071177, avg = 0.037846, mid = 0.040507
In stage 2 >> min = 0.000000, max = 0.011831, avg = 0.007812, mid = 0.006954
In stage 3 >> min = 0.000000, max = 0.006838, avg = 0.004571, mid = 0.004138
In all stages >> min = 0.000000, max = 0.082282, avg = 0.050229, mid = 0.051321
In system >> min = 0.020062, max = 0.231577, avg = 0.133734, mid = 0.139131
```

Рисунок 4.5 – Статистика времени обработки заявок

Можно увидеть, что очереди с наибольшим временем нахождения в них заявки — первая и вторая. Для первой очереди такой результат объясняется тем, что она заполняется генератором заранее, и последняя заявка находится в ней до того момента, пока все предшествующие ей не будут обработаны первой лентой. Это подтверждает и среднее время, проведенное заявкой в первой очереди, которое приблизительно равно половине от максимального.

Для второй очереди наибольшее максимальное время нахождения в ней заявки связано со сложностью работы соответсвующей ей ленты. Она приводит матрицы к верхнетреугольному виду, выполняя преобразования над строками и переставляя их, на что затрачивается большое количество операций и, соответсвенно, время.

Вывод

В результате эксперимента было получено, что использование конвейрной обработки эффективнее последовательной реализации по времени работы при количестве матриц, равном 10, в 1.75 раз, а при количестве матриц, равном 50, уже в 1.88 раз. Следовательно, преимущество конвейерного алгоритма обработки матриц над последовательным растет с увеличением количества задач, в данном случае матриц.

Заключение

В результате выполнения лабораторной работы цель достигнута: получен навык организации асинхронного взаимодействия между потоками на примере моделирования конвейера.

В ходе выполнения данной работы были решены все задачи:

1) изучены основы конвейерной обработки данных;
2) изучены алгоритмы возведения матрицы в квадрат, приведения матрицы к верхнетреугольному виду и нахождения определителя матрицы;
3) разработаны параллельная и последовательная версии конвейера с тремя стадиями обработки;
4) реализованы параллельная и последовательная версии конвейера с тремя стадиями обработки;
5) проведен сравнительный анализ времени работы последовательной и параллельной реализаций конвейера при различном количестве задач;
6) собрана статистика времени ожидания заявок (максимальное, минимальное, среднее, медианное значение) в каждой из очередей (лент) конвейера по отдельности, во всех очередях и времени обработки в системе.

В результате лабораторной работы можно сделать вывод, что использование конвейрной обработки эффективнее последовательной реализации по времени работы, причем преимущество конвейерного алгоритма обработки матриц над последовательным растет с увеличением количества задач.

Список использованных источников

1. Принципы конвейерной технологии [Электронный ресурс]. Режим доступа: https://www.sites.google.com/site/shoradimon/18-principy-konvejernoj-tehnologii (дата обращения: 07.12.2022).

2. Нахождение определителя матрицы методом Гаусса [Электронный ресурс]. Режим доступа: https://studwork.org/spravochnik/matematika/matricy/nahojdenie-opredelitelya-matricy-metodom-gaussa (дата обращения: 07.12.2022).

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: 29.10.2022).

4. Date and time utilities [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/cpp/chrono (дата обращения: 29.10.2022).

5. Документация по Clion [Электронный ресурс]. Режим доступа: https://www.jetbrains.com/help/clion/viewing-inline-documentation.html (дата обращения: 13.07.2022).

6. Лутц, Марк. Изучаем Python, том 1, 5-е изд. Пер. с англ. — СПб.: ООО “Диалектика”, 2019 — С. 832.

7. Windows 10 [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/windows/ (дата обращения: 20.09.2022).