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 ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе № 5

по курсу «Анализ алгоритмов»

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

Студент ИУ7-55Б _____ И. Д. Половинкин
(Подпись, дата)

Преподаватель _____ Л. Л. Волкова
(Подпись, дата)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Параллельные вычисления позволяют увеличить скорость выполнения программ. Конвейерная обработка данных является популярным приемом при работе с параллельностью. Она позволяет на каждой следующей «линии» конвейера использовать данные, полученные с предыдущего этапа.

Конвейер — способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности.

Целью данной лабораторной работы является изучение принципов конвейерной обработки данных.

Для достижения поставленной цели необходимо выполнить следующие задачи:

— исследовать основы конвейерной обработки данных;

— привести схемы алгоритмов, используемых для конвейерной и линейной обработок данных;

— определить средства программной реализации;

— провести модульное тестирование;

— провести сравнительный анализ времени работы алгоритмов;

— описать и обосновать полученные результаты.

1 Аналитический раздел

В данном разделе представлено описание сути конвертерной обработки данных и используемый алгоритм обработки матрицы для конвейерных вычислений.

1.1 Конвейерная обработка данных

Конвейерная обработка данных — способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности за счет увеличения числа инструкций, выполняемых в единицу времени — эксплуатация параллелизма на уровне инструкций [1].

Конвейерную обработку используют для совмещения этапов выполнения разных команд. Производительность возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько команд. Такая обработка данных в общем случае основана на разделении функции на более мелкие части, называемые лентами, и выделении для каждой из них отдельного блока аппаратуры. Таким образом, обработку любой машинной команды можно разделить на несколько этапов (лент), организовав передачу данных от одного этапа к следующему.

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, не конвейерной схемой.

1.2 Описание алгоритмов конвейерной обработки

В данной лабораторной работе на основе конвейерной обработке данных будет обрабатываться матрица. В качестве алгоритмов на каждую из трех лент были выбраны следующие действия.

1. Нахождение наименьшего элемента в матрице.
2. Запись в каждую ячейку матрицы остатка от деления текущего элемента на минимальный элемент.

3. Нахождение суммы элементов полученной матрицы.

Вывод

В разделе было рассмотрено понятие конвейрной обработки данных, а также описаны алгоритмы для обработки матрицы на каждой из трех лент конвейера.

2 Конструкторский раздел

В данном разделе приводятся схемы линейной и конвейерной реализации алгоритмов обработки матриц.

2.1 Алгоритмы обработки матриц

2.1.1 Линейный алгоритм

На рисунке 2.1 представлен линейный алгоритм обработки матриц.

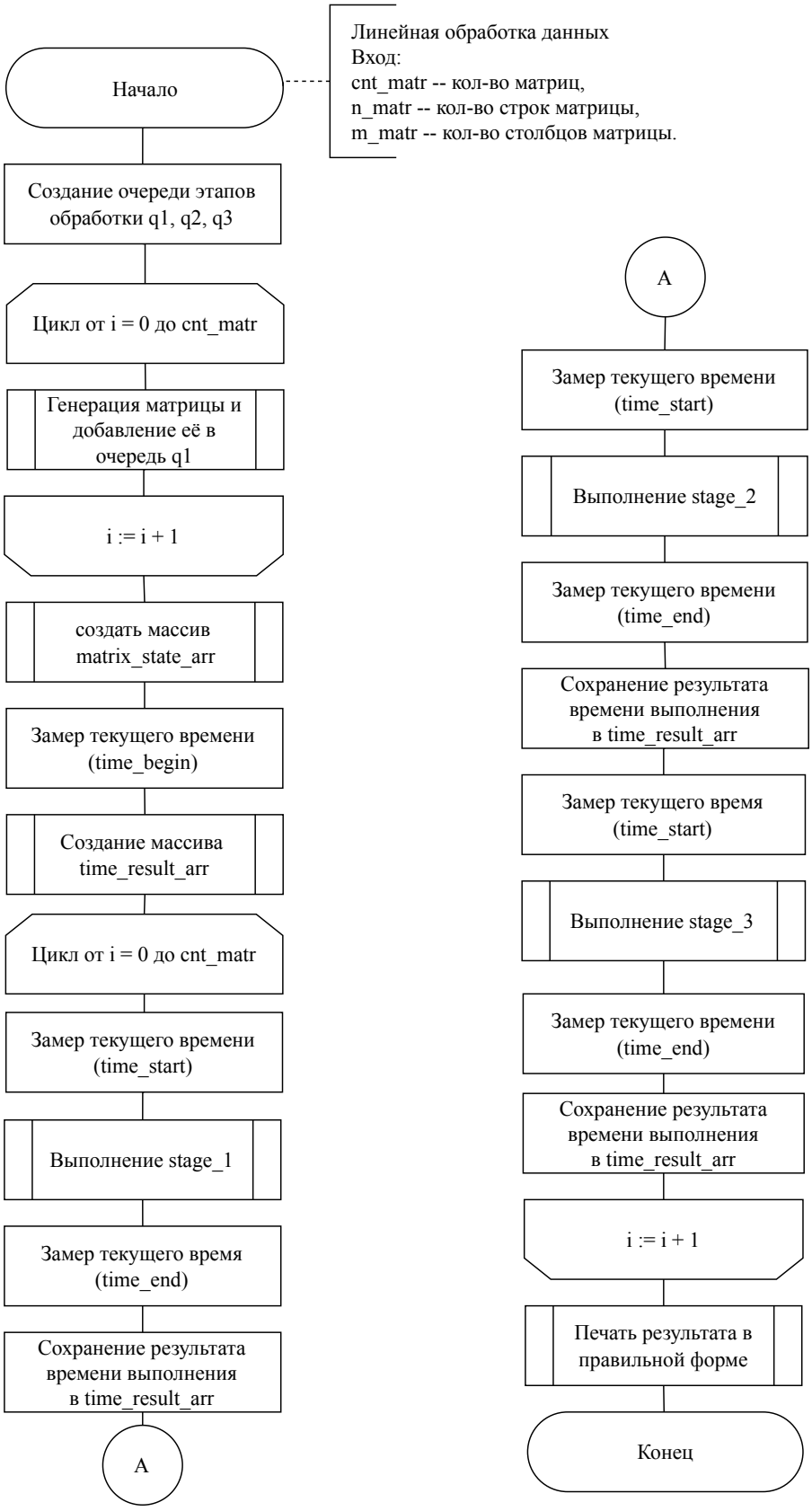

Рисунок 2.1 – Алгоритм линейной обработки матриц

2.1.2 Алгоритм конвейерной обработки матриц

На рисунке 2.2 представлен алгоритм конвейерной обработки матриц.

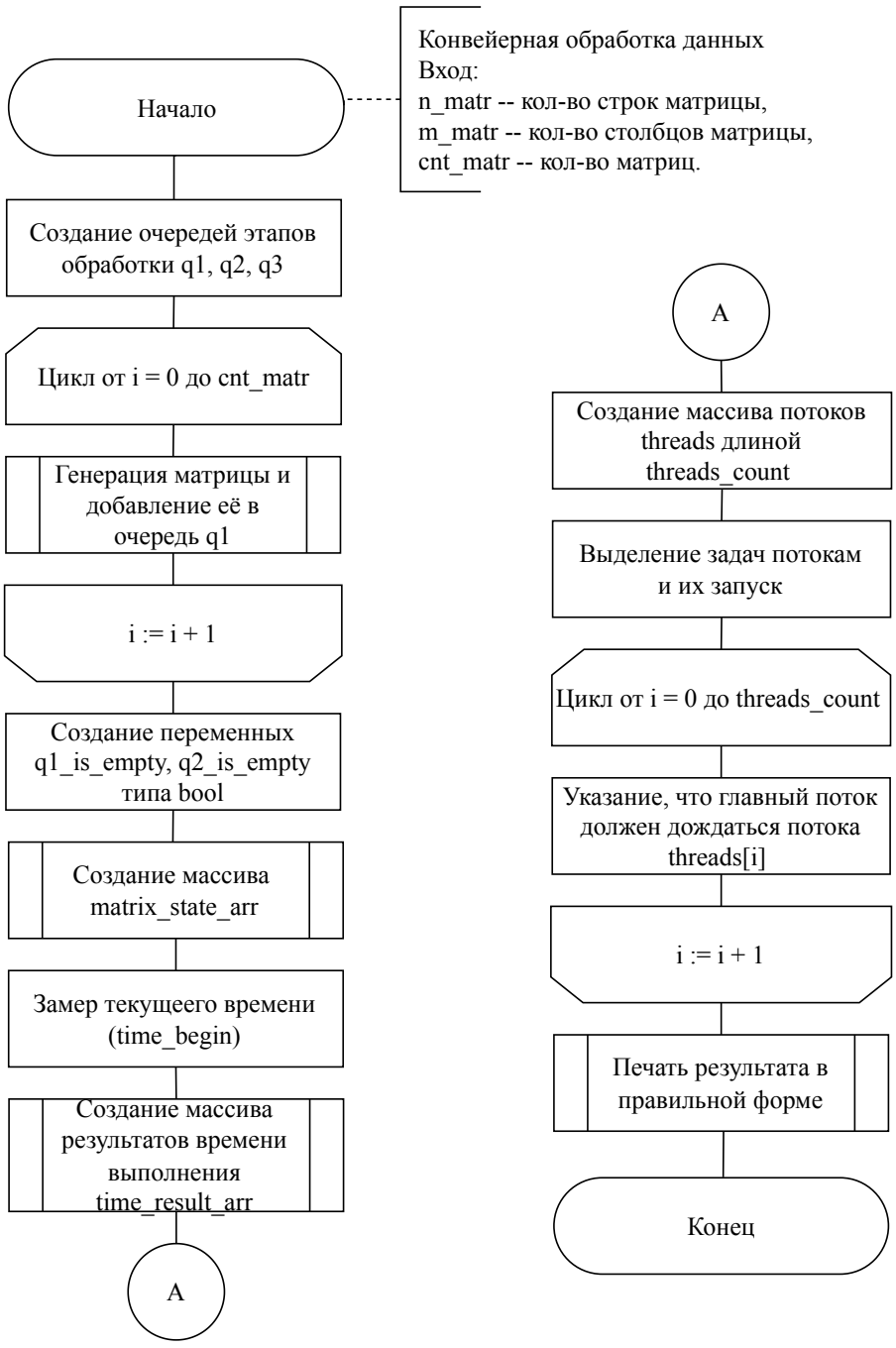

Рисунок 2.2 – Алгоритм конвейерной обработки матриц

2.1.3 Алгоритм конвейерной обработки матрицы 1-ой ленты

На рисунке 2.3 представлен алгоритм конвейерной обработки матриц 1-ой ленты.

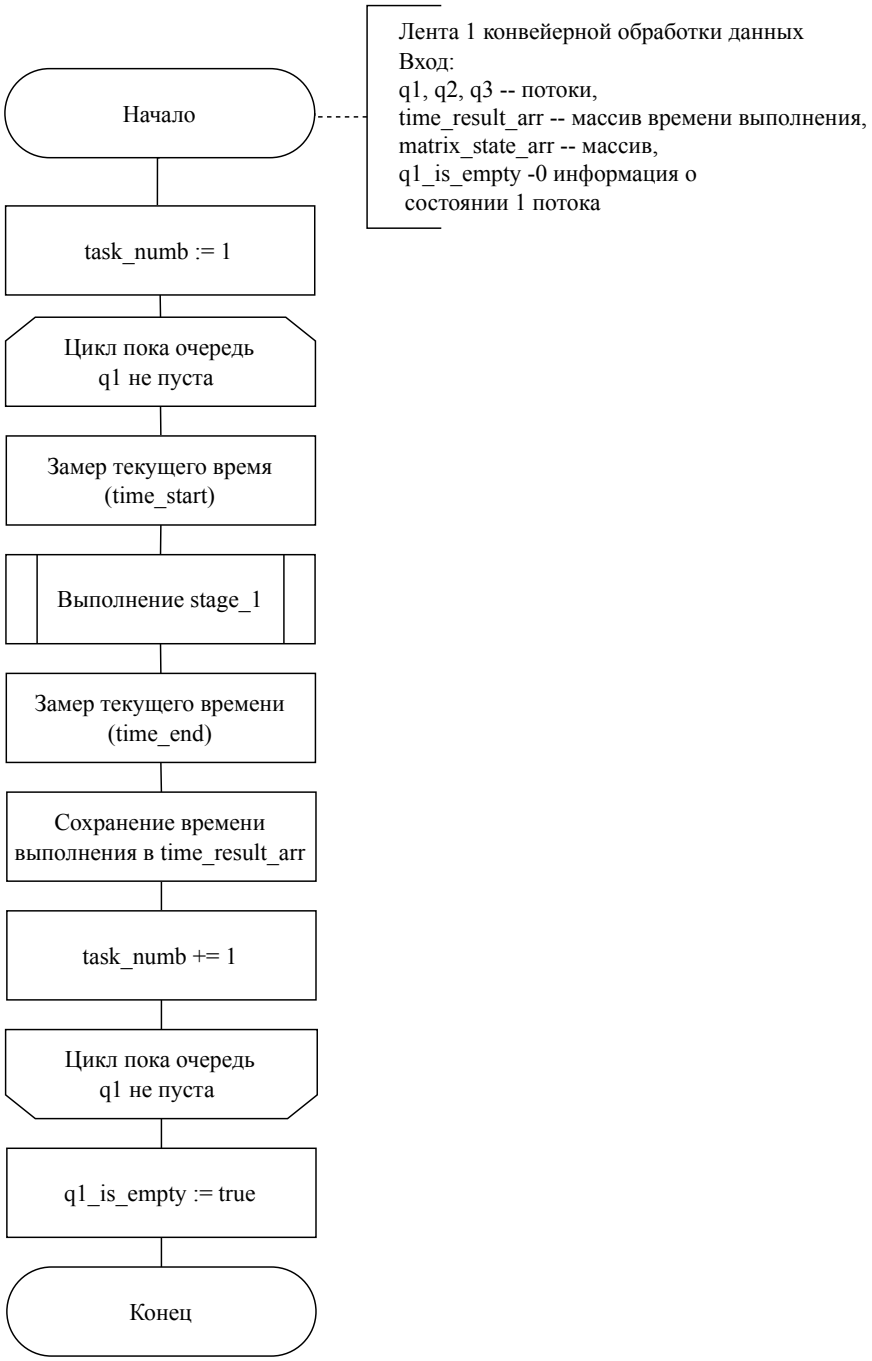

Рисунок 2.3 – Алгоритм конвейерной обработки матрицы 1-ой ленты

2.1.4 Алгоритм конвейерной обработки матрицы 2-ой ленты

На рисунке 2.4 представлен алгоритм конвейерной обработки матриц 2-ой ленты.

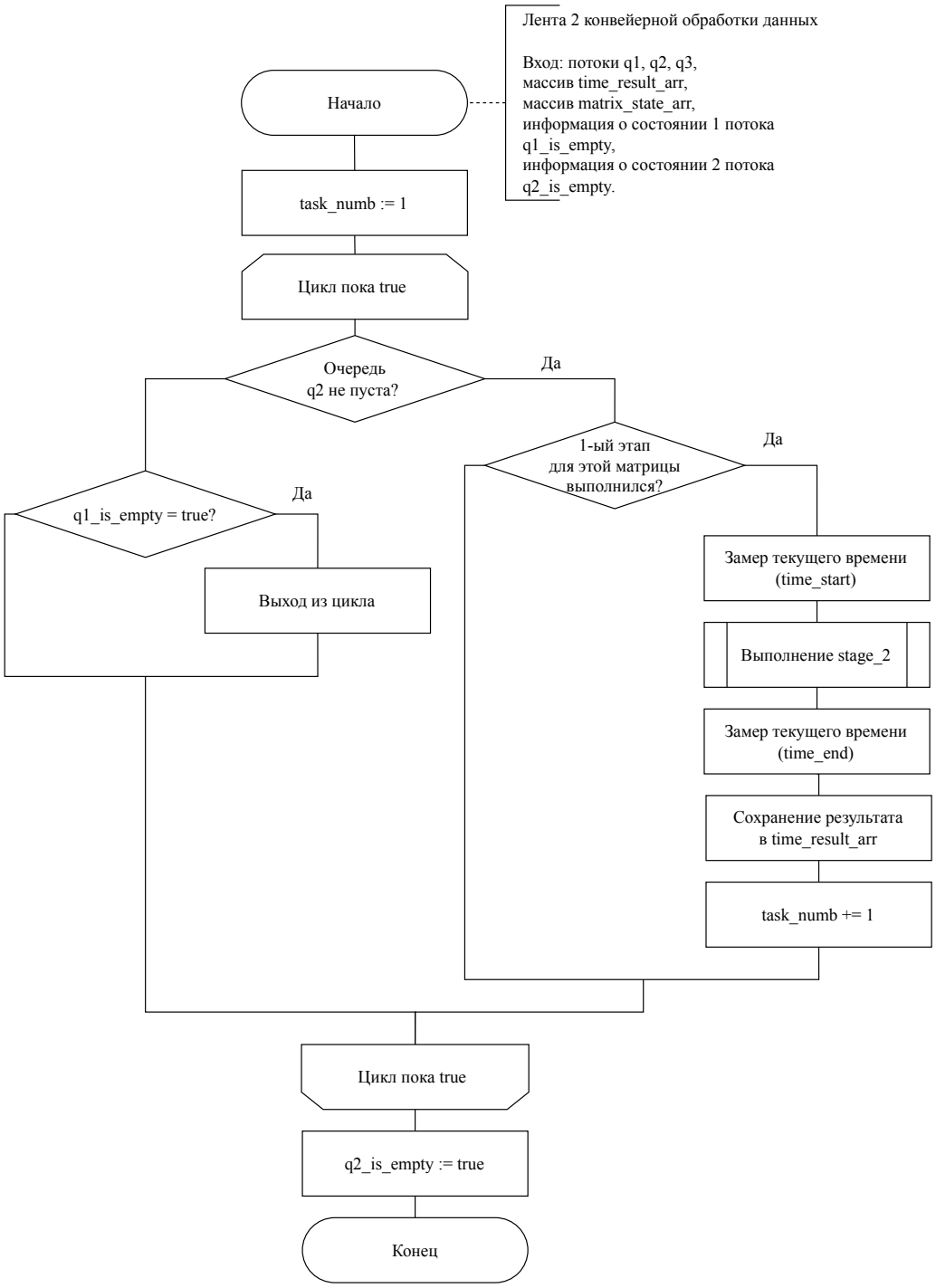

Рисунок 2.4 – Алгоритм конвейерной обработки матрицы 2-ой ленты

2.1.5 Алгоритм конвейерной обработки матрицы 3-ей ленты

На рисунке 2.5 представлен алгоритм конвейерной обработки матриц 3-ей ленты.

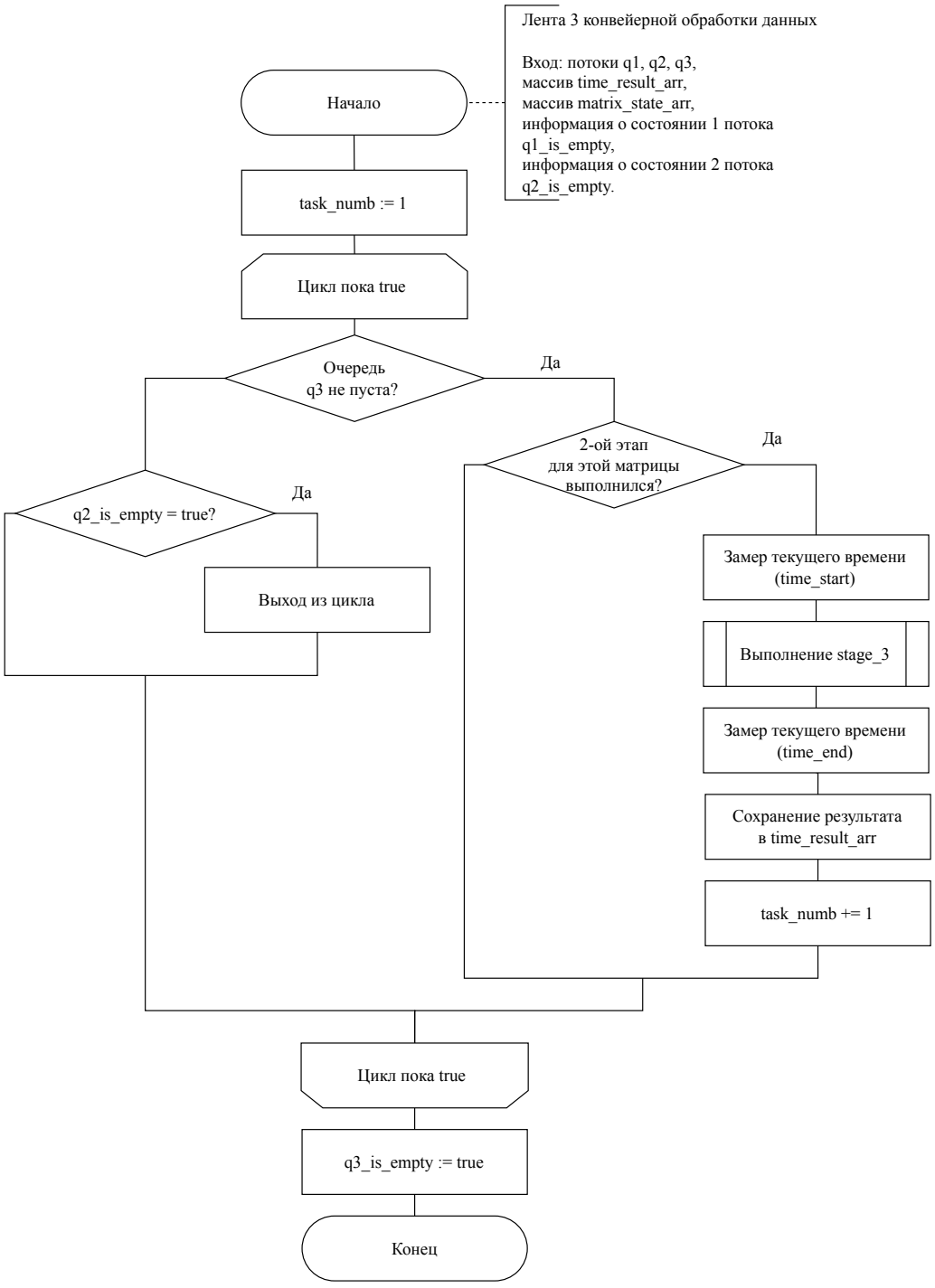

Рисунок 2.5 – Алгоритм конвейерной обработки матрицы 3-ой ленты

2.1.6 Алгоритмы этапов обработки матрицы

На рисунке 2.6 представлены схемы этапов обработки матрицы.

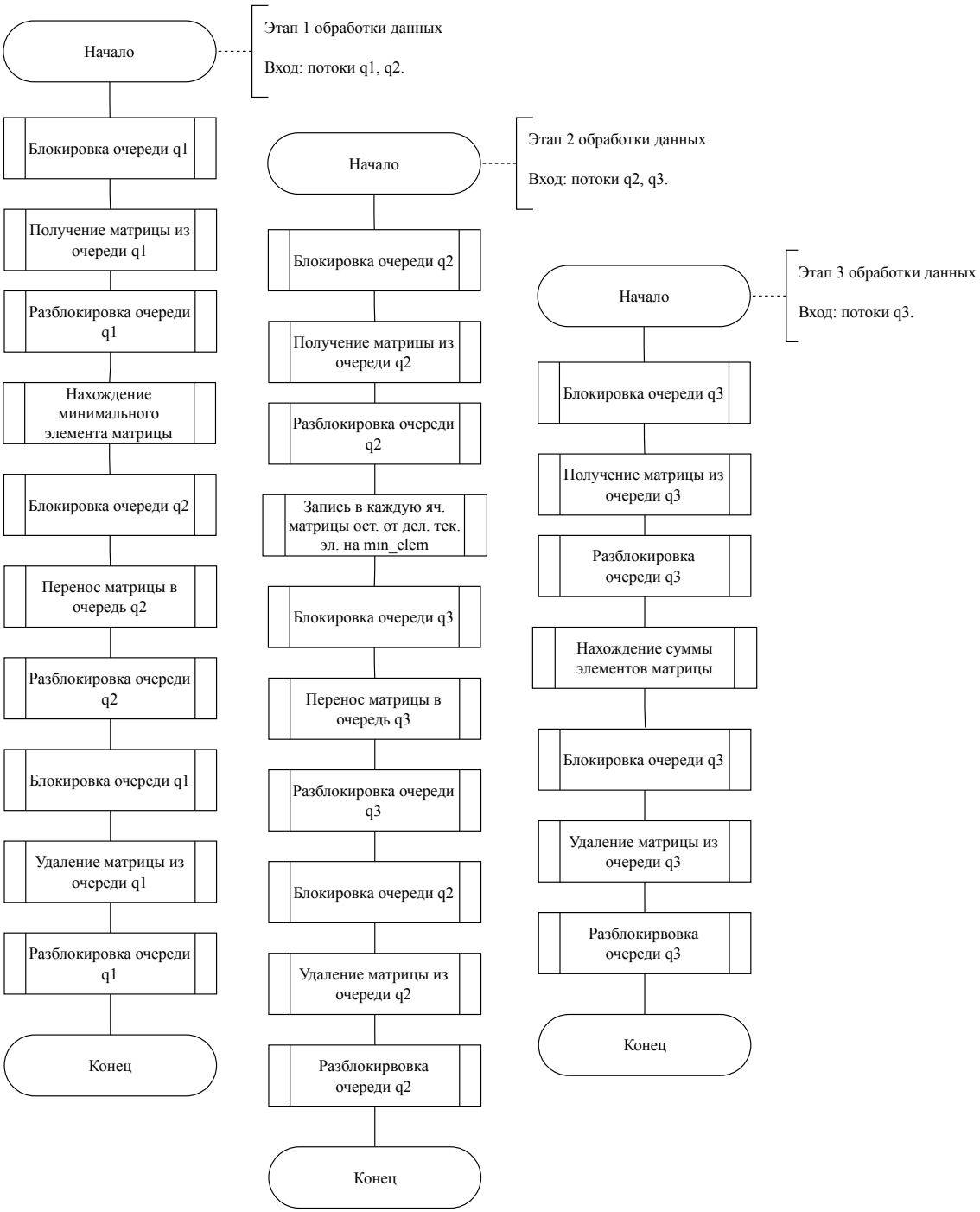

Рисунок 2.6 – Схемы этапов обработки матрицы

Вывод

В данном разделе были приведены схемы методов обработки матриц (линейного и конвейерного).

3 Технологический раздел

В данном разделе приводятся требования к программному обеспечению, средства реализации, модули программы, а также функциональные тесты.

3.1 Требования к программному обеспечению

Программа должна отвечать следующим требованиям:

– на вход программе задается количество строк и столбцов матрицы, большее 0;

– все элементы матрицы имеют тип int;

– на выходе программа выдает таблицу с номерами матриц, номерами этапов (лент) её обработки, временем начала обработки текущей матрицы на текущей ленте, временем окончания обработки текущей матрицы на текущей ленте.

3.2 Средства реализации

В работе для реализации алгоритмов был выбран язык программирования C++ [2], поскольку он предоставляет необходимую структуру данных для организации очереди std::queue [3] и средство взаимного исключения потоков std::mutex [4], а для создания потока используется std::thread [5]. Для замера времени работы алгоритмов используется функция std::chrono::system_clock_now() [6].

3.3 Модули программы

На листинге 3.1 представлена функция конвейерной обработки матрицы.

Листинг 3.1 – Функция конвейерной обработки матрицы

```
void parallel_processing(int n_matr, int m_matr,
    int cnt_matr, bool matr_is_print, bool compare_time)
{
    std::queue<matrix_t> q1, q2, q3;
    std::mutex m;

    for (int i = 0; i < cnt_matr; i++) {
        matrix_t matr = generate_matrix(n_matr, m_matr);
        q1.push(matr);

        if (matr_is_print && i == cnt_matr - 1)
        {
            m.lock();
            printf("start matrix:\n");
            print_matrix(matr);
            m.unlock();
        }
    }

    bool q1_is_empty = false;
    bool q2_is_empty = false;

    std::vector<matrix_state_t> matrix_state_arr;
    init_matrix_state_arr(matrix_state_arr, cnt_matr);

    std::chrono::time_point<std::chrono::system_clock> time_begin =
    std::chrono::system_clock::now();

    std::vector<res_time_t> time_result_arr;
    init_time_result_arr(time_result_arr, time_begin, cnt_matr, THREADS_COUNT);

    std::thread threads[THREADS_COUNT];

    threads[0] = std::thread(parallel_stage_1, std::ref(q1), std::ref(q2),
        std::ref(time_result_arr), std::ref(matrix_state_arr),
        std::ref(q1_is_empty));
    threads[1] = std::thread(parallel_stage_2, std::ref(q2),
        std::ref(q3), std::ref(time_result_arr),
        std::ref(matrix_state_arr), std::ref(q1_is_empty),
        std::ref(q2_is_empty));
    threads[2] = std::thread(parallel_stage_3, std::ref(q3),
        std::ref(time_result_arr), std::ref(matrix_state_arr),
        sstd::ref(q2_is_empty), cnt_matr, matr_is_print);

    for (int i = 0; i < THREADS_COUNT; i++) {
        threads[i].join();
    }

    if (compare_time) {
        printf("     %4d      %s|%s     %4d      %s|%s   %.6f  \n",
            n_matr, PURPLE, BASE_COLOR,
            cnt_matr, PURPLE, BASE_COLOR,
            time_result_arr[cnt_matr - 1].end);
    }
    else {
        print_res_time(time_result_arr, cnt_matr * THREADS_COUNT);
    }
}
```

На листинге 3.2 представлена функция 1-ой ленты конвейерной обработки матрицы.

Листинг 3.2 – Функция 1-ой ленты конвейерной обработки матрицы

```
void stage_1(std::queue<matrix_t> &q1, std::queue<matrix_t> &q2)
{
    std::mutex m;

    m.lock();
    matrix_t matr = q1.front();
    m.unlock();

    matr.min_elem = get_min_elem(matr);

    m.lock();
    q2.push(matr);
    m.unlock();

    m.lock();
    q1.pop();
    m.unlock();
}
```

На листинге 3.3 представлена функция 2-ой ленты конвейерной обработки матрицы.

Листинг 3.3 – Функция 2-ой ленты конвейерной обработки матрицы

```
void stage_2(std::queue<matrix_t> &q2, std::queue<matrix_t> &q3)
{
    std::mutex m;

    m.lock();
    matrix_t matr = q2.front();
    m.unlock();

    mod_by_min_elem(matr);

    m.lock();
    q3.push(matr);
    m.unlock();

    m.lock();
    q2.pop();
    m.unlock();
}
```

На листинге 3.4 представлена функция 3-ей ленты конвейерной обработки матрицы.

Листинг 3.4 – Функция 3-ей ленты конвейерной обработки матрицы

```
void stage_3(std::queue<matrix_t> &q3, int task_num, int cnt_matr, bool matr_is_print)
{
    std::mutex m;

    m.lock();
    matrix_t matr = q3.front();
    m.unlock();

    matr.sum_elem = get_sum_elements(matr);

    if (matr_is_print && task_num == cnt_matr)
    {
        printf("\nmin_elem = %d\n\nmatrix after 2 step:\n", matr.min_elem);
        print_matrix(matr);
        printf("\nsum_elem = %d\n\n", matr.sum_elem);
    }

    m.lock();
    q3.pop();
    m.unlock();
}
```

На листинге 3.5 представлена функция линейной обработки матрицы.

Листинг 3.5 – Функция линейной обработки матрицы

```
void linear_processing(int n_matr, int m_matr, int cnt_matr, bool matr_is_print, bool compare_time)
{
    std::queue<matrix_t> q1, q2, q3;
    std::mutex m;

    for (int i = 0; i < cnt_matr; i++) {
        matrix_t matr = generate_matrix(n_matr, m_matr);
        q1.push(matr);

        if (matr_is_print && i == cnt_matr - 1) {
            m.lock();
            printf("start matrix:\n");
            print_matrix(matr);
            m.unlock();
        }
    }

    std::vector<matrix_state_t> matrix_state_arr;
    init_matrix_state_arr(matrix_state_arr, cnt_matr);

    std::chrono::time_point<std::chrono::system_clock> time_start, time_end,
    time_begin = std::chrono::system_clock::now();

    std::vector<res_time_t> time_result_arr;
    init_time_result_arr(time_result_arr, time_begin, cnt_matr, THREADS_COUNT);

    for (int i = 0; i < cnt_matr; i++)
    {
        time_start = std::chrono::system_clock::now();
        stage_1(std::ref(q1), std::ref(q2));
        time_end = std::chrono::system_clock::now();

        save_result(time_result_arr, time_start, time_end, time_begin, i + 1, 1);

        time_start = std::chrono::system_clock::now();
        stage_2(std::ref(q2), std::ref(q3));
        time_end = std::chrono::system_clock::now();

        save_result(time_result_arr, time_start, time_end, time_begin, i + 1, 2);

        time_start = std::chrono::system_clock::now();
        stage_3(std::ref(q3), i + 1, cnt_matr, matr_is_print);
        time_end = std::chrono::system_clock::now();

        save_result(time_result_arr, time_start, time_end, time_begin, i + 1, 3);
    }

    if (compare_time) {
        printf("     %4d      %s|%s     %4d      %s|%s   %.6f  \n",
            n_matr, PURPLE, BASE_COLOR,
            cnt_matr, PURPLE, BASE_COLOR,
            time_result_arr[cnt_matr - 1].end);
    }
    else {
        print_res_time(time_result_arr, cnt_matr * THREADS_COUNT);
    }
}
```

На листинге 3.6 представлена функция 1-ой ленты линейной обработки матрицы.

Листинг 3.6 – Функция 1-ой ленты линейной обработки матрицы

```
void stage_1(std::queue<matrix_t> &q1, std::queue<matrix_t> &q2)
{
    std::mutex m;

    m.lock();
    matrix_t matr = q1.front();
    m.unlock();

    matr.min_elem = get_min_elem(matr);

    m.lock();
    q2.push(matr);
    m.unlock();

    m.lock();
    q1.pop();
    m.unlock();
}
```

На листинге 3.7 представлена функция 2-ой ленты линейной обработки матрицы.

Листинг 3.7 – Функция 2-ой ленты линейной обработки матрицы

```
void stage_2(std::queue<matrix_t> &q2, std::queue<matrix_t> &q3)
{
    std::mutex m;

    m.lock();
    matrix_t matr = q2.front();
    m.unlock();

    mod_by_min_elem(matr);

    m.lock();
    q3.push(matr);
    m.unlock();

    m.lock();
    q2.pop();
    m.unlock();
}
```

На листинге 3.8 представлена функция 3-ей ленты линейной обработки матрицы.

Листинг 3.8 – Функция 3-ей ленты линейной обработки матрицы

```
void stage_3(std::queue<matrix_t> &q3, int task_num, int cnt_matr, bool matr_is_print)
{
    std::mutex m;

    m.lock();
    matrix_t matr = q3.front();
    m.unlock();

    matr.sum_elem = get_sum_elements(matr);

    if (matr_is_print && task_num == cnt_matr)
    {
        printf("\nmin_elem = %d\n\nmatrix after 2 step:\n", matr.min_elem);
        print_matrix(matr);
        printf("\nsum_elem = %d\n\n", matr.sum_elem);
    }

    m.lock();
    q3.pop();
    m.unlock();
}
```

3.4 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для конвейерного и ленейного алгоритмов обработки матриц. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Строк	Столбцов	Метод обр.	Алгоритм	Ожидаемый результат
0	10	10	Конвейерный	Сообщение об ошибке
k	10	10	Конвейерный	Сообщение об ошибке
10	0	10	Конвейерный	Сообщение об ошибке
10	k	10	Конвейерный	Сообщение об ошибке
10	10	-5	Конвейерный	Сообщение об ошибке
10	10	k	Конвейерный	Сообщение об ошибке
100	100	20	Конвейерный	Вывод результ. таблички
100	100	20	Линейный	Вывод результ. таблички
50	100	100	Линейный	Вывод результ. таблички

Вывод

В данном разделе были разработаны алгоритмы для конвейерного и линейного алгоритмов обработки матриц, проведено тестирование, описаны средства реализации и требования к программе.

4 Исследовательский раздел

4.1 Технические характеристики

Технические характеристии устройства, на котором выполнялось тестирование:

- операционная система: Windows 10 Pro;
- память: 8 GiB;
- процессор: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz.

Тестирование проводилось на ноутбуке, который был подключен к сети питания. Во время проведения тестирования ноутбук был нагружен только встроенными приложениями окружения, самим окружением и системой тестирования.

4.2 Время выполнения алгоритмов

Результаты замеров времени работы алгоритмов обработки матриц для конвейерной и ленейной реализаций представлены на рисунках 4.1 – 4.2. Замеры времени проводились в секундах и усреднялись для каждого набора одинаковых экспериментов.

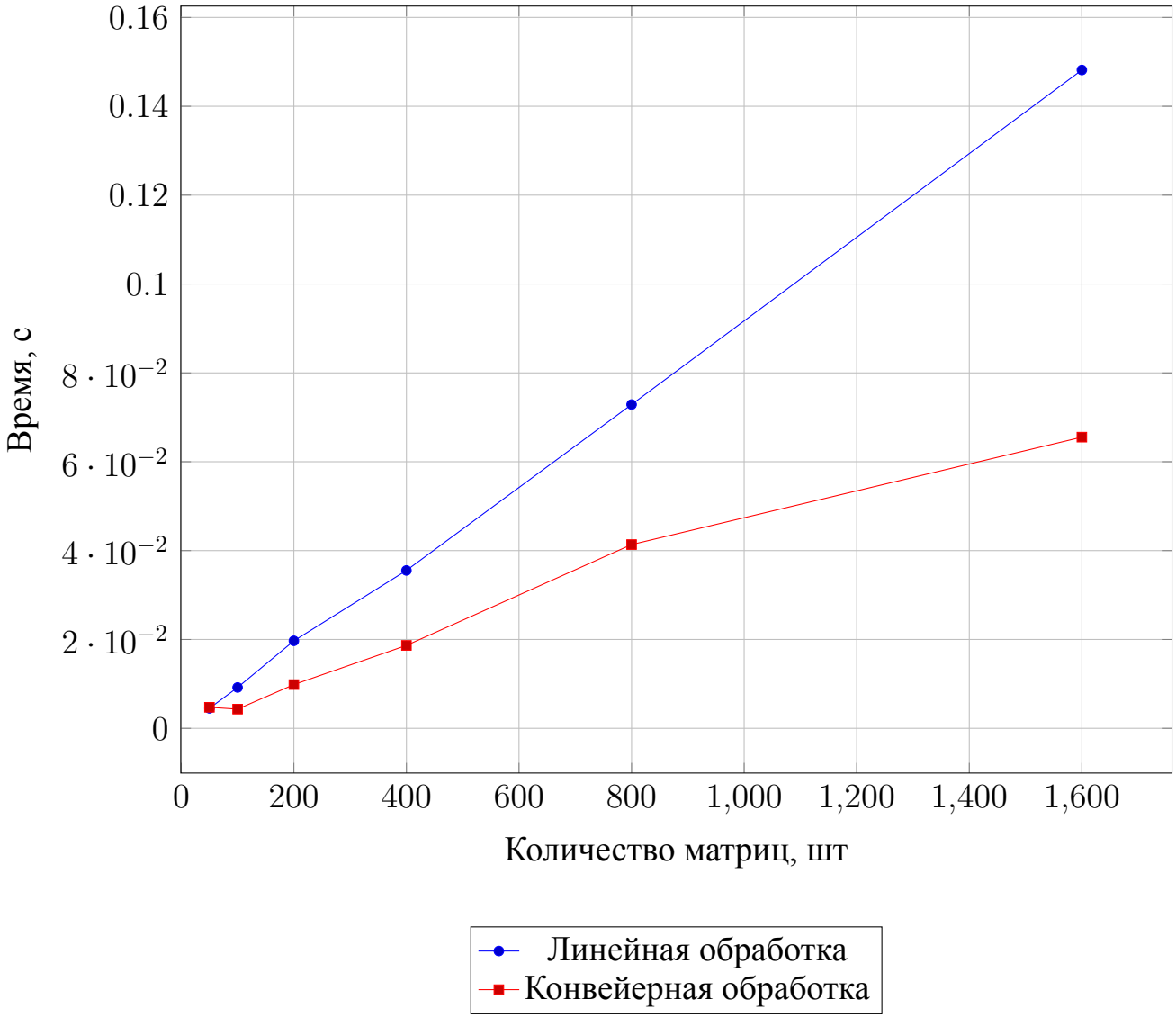

Рисунок 4.1 – Зависимость времени обработки алгоритма от количества матриц

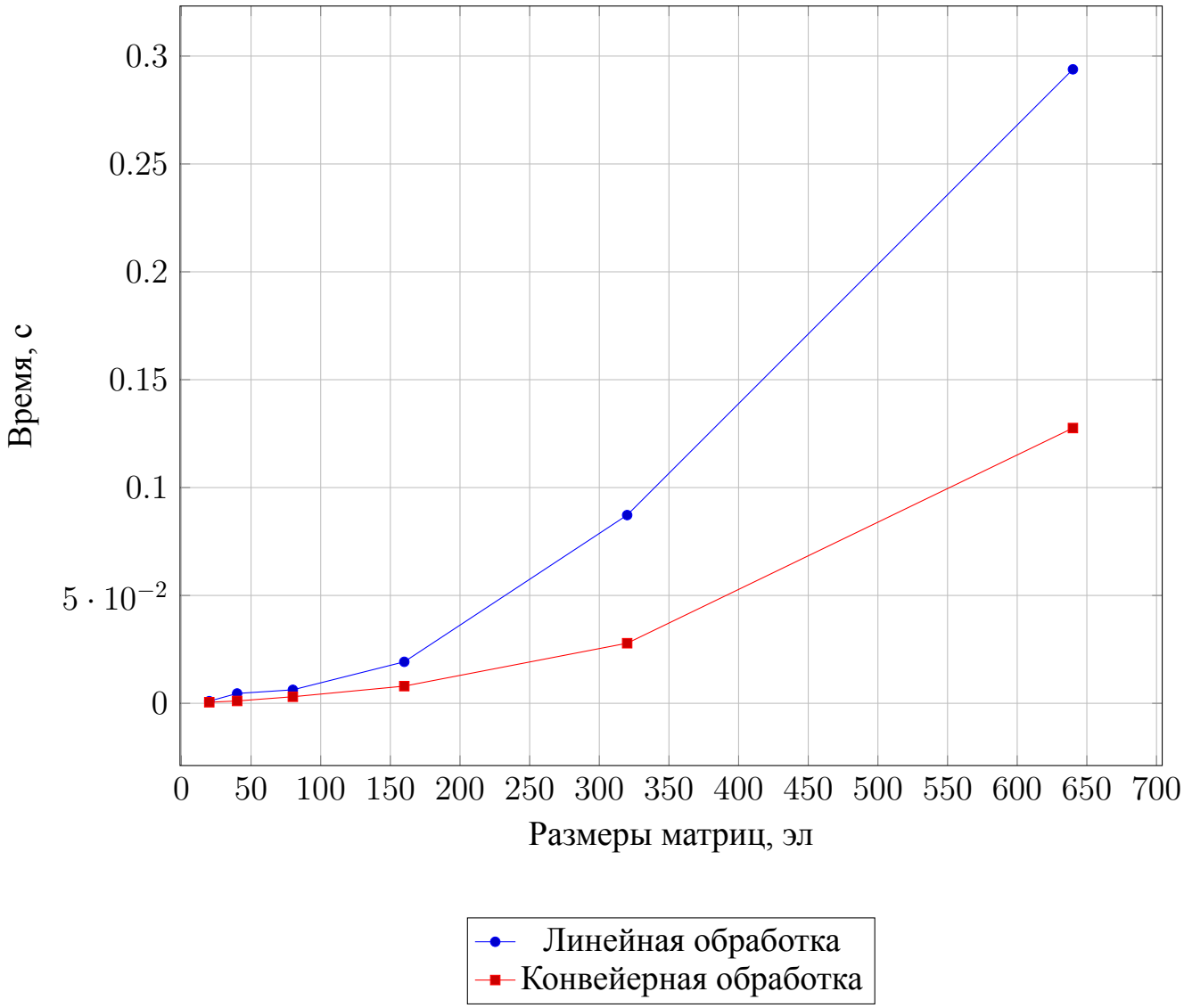

Рисунок 4.2 – Зависимость времени обработки алгоритма от размеров матриц

Вывод

В результате замеров времени было установлено, что конвейерная реализация обработки лучше линейной при большом количестве матриц (в 2.5 раза при 400 матрицах, в 2.6 раза при 800 и в 2.7 при 1600). Так же конвейерная обработка показала себя лучше при увеличении размеров обрабатываемых матриц (в 2.8 раза при размере матриц 160x160, в 2.9 раза при размере 320x320 и в 2.9 раза при матрицах 640x640). Значит при большом количестве обрабатываемых матриц, а так же при матрицах большого размера стоит использовать конвейерную реализацию обработки, а не линейную.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были рассмотрены способы организации вычислений: линейный и конвейерный. Для организации конвейерных вычислений был предложен алгоритм обработки матриц и её элементов.

Экспериментально подтверждено различие во временной эффективности конвейерной и линейной реализаций обработки матриц. При большом количестве обрабатываемых матриц, а так же при матрицах большого размера стоит использовать конвейерную реализацию обработки, а не линейную (при 1600 матриц конвейерная быстрее в 2.7 раза, а при матрицах 640x640 быстрее в 2.9 раза).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Конвейерная обработка данных [Электронный ресурс]. – Режим доступа, URL: https://studfile.net/preview/1083252/page:25/ (дата обращения: 14.02.2024)

2. C++17 [Электронный ресурс]. – Режим доступа, URL: https://en.cppreference.com/w/cpp/17 (дата обращения: 15.02.2024)

3. std::queue [Электронный ресурс]. – Режим доступа, URL: https://en.cppreference.com/w/cpp/container/queue (дата обращения: 15.02.2024)

4. std::mutex [Электронный ресурс]. – Режим доступа, URL: https://en.cppreference.com/w/cpp/thread/mutex (дата обращения: 15.02.2024)

5. std::mutex [Электронный ресурс]. – Режим доступа, URL: https://en.cppreference.com/w/cpp/thread/thread (дата обращения: 15.02.2024)

6. std::chrono::system_clock::now [Электронный ресурс]. – Режим доступа, URL: https://en.cppreference.com/w/cpp/chrono/system_clock/now (дата обращения: 15.02.2024)