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»

Лабораторная работа №5
по дисциплине "Анализ Алгоритмов"

Тема Конвейерная обработка данных

Студент Ковалец К. Э.

Группа ИУ7-53Б

Преподаватель Волкова Л. Л.

Москва — 2021 г.

Содержание

Введение

Для увеличения скорости выполнения программ используют параллельные вычисления. Конвейерная обработка данных является популярным приемом при работе с параллельностью. Она позволяет на каждой следующей «линии» конвейера использовать данные, полученные с предыдушего этапа.

Конвейер — способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности (эксплуатация параллелизма на уровне инструкций).

Целью данной лабораторной работы является изучение принципов конвейрной обработки данных.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- исследовать основы конвейрной обработки данных;
- привести схемы алгоритмов, используемых для конвейрной и линейной обработок данных;
- описать используемые структуры данных;
- описать структуру разрабатываемого ПО;
- определить средства программной реализации;
- провести сравнительный анализ времени работы алгоритмов;
- провести модульное тестирование;
- описать и обосновать полученные результаты в отчете о выполненной лабораторной работе.

1 Аналитическая часть

В этом разделе будет представлено описание сути конвейрной обработки данных и используемых алгоритмов.

1.1 Описание конвейерной обработки данных

Конвейер [1] — 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.

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

1.2 Описание алгоритмов

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

- Найти наименьший элемент в матрице min_elem.
- Записать в каждую ячейку матрицы остаток от деления текущего элемента на min_elem.
- Найти сумму элементов полученной матрицы.

1.3 Вывод

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

На вход программе будет поступать кол-во матриц и её размер (кол-во строк и столбцов). При попытке задать некорректные данные, будет выдано сообщение об ошибке. Реализуемое ПО будет давать возможность выбрать метод обработки данных (конвейрный или линейный) и вывести для него результат вычисления, а также возможность произвести сравнение алгоритмов по затраченному времени.

2 Конструкторская часть

В данном разделе будут приведены схемы конвейерной и линейной реализаций алгоритмов обработки матриц, приведено описание используемых типов данных, а также описана структура ПО.

2.1 Схемы алгоритмов

На рис. 2.1 - 2.6 приведены схемы линейной и конвейерной реализаций алгоритмов обработки матрицы, схема трёх лент для конвейерной обработки матрицы, а также схемы реализаций этапов обработки матроицы.

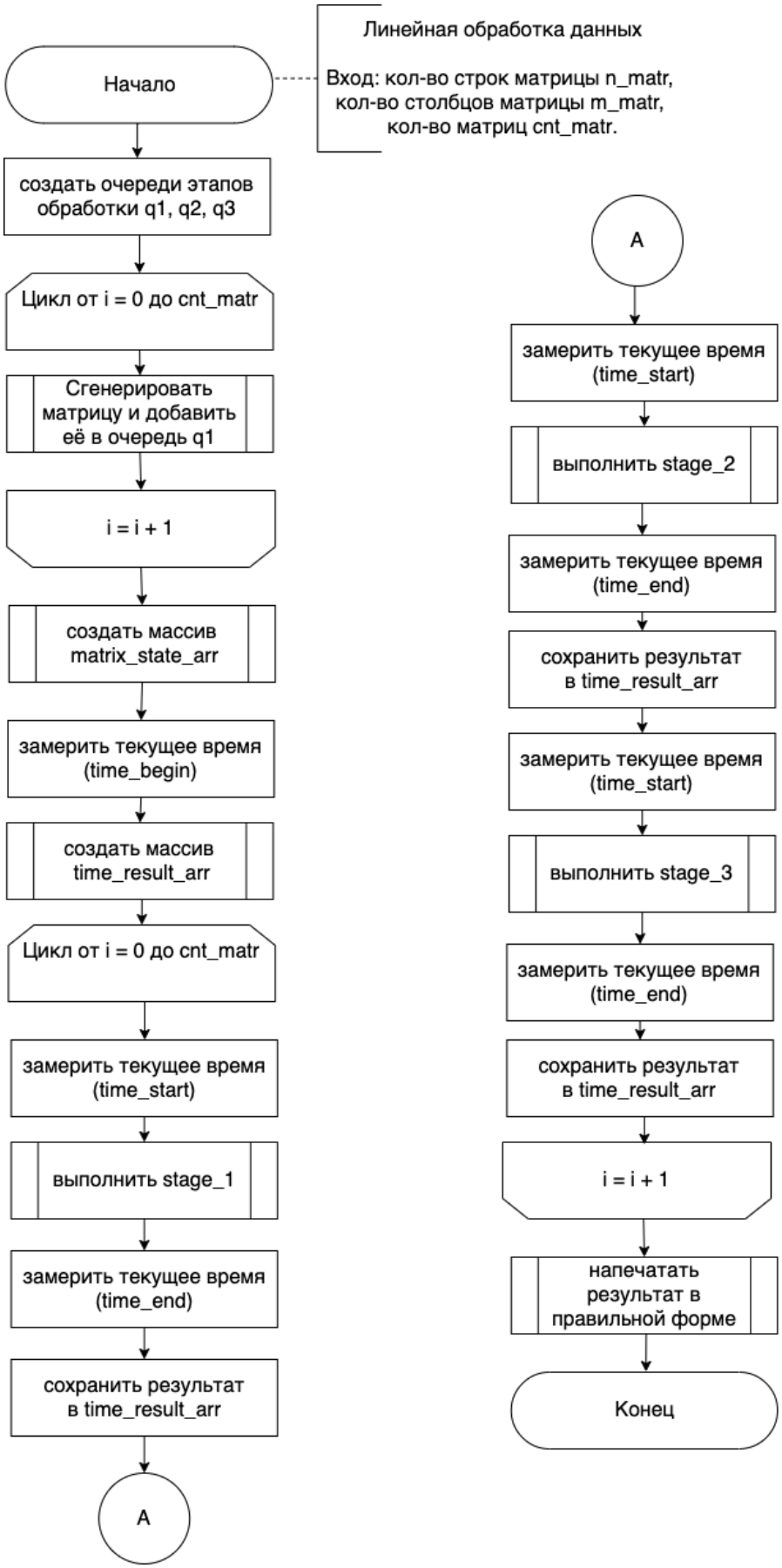

Рисунок 2.1 – Схема алгоритма линейной обработки матроицы

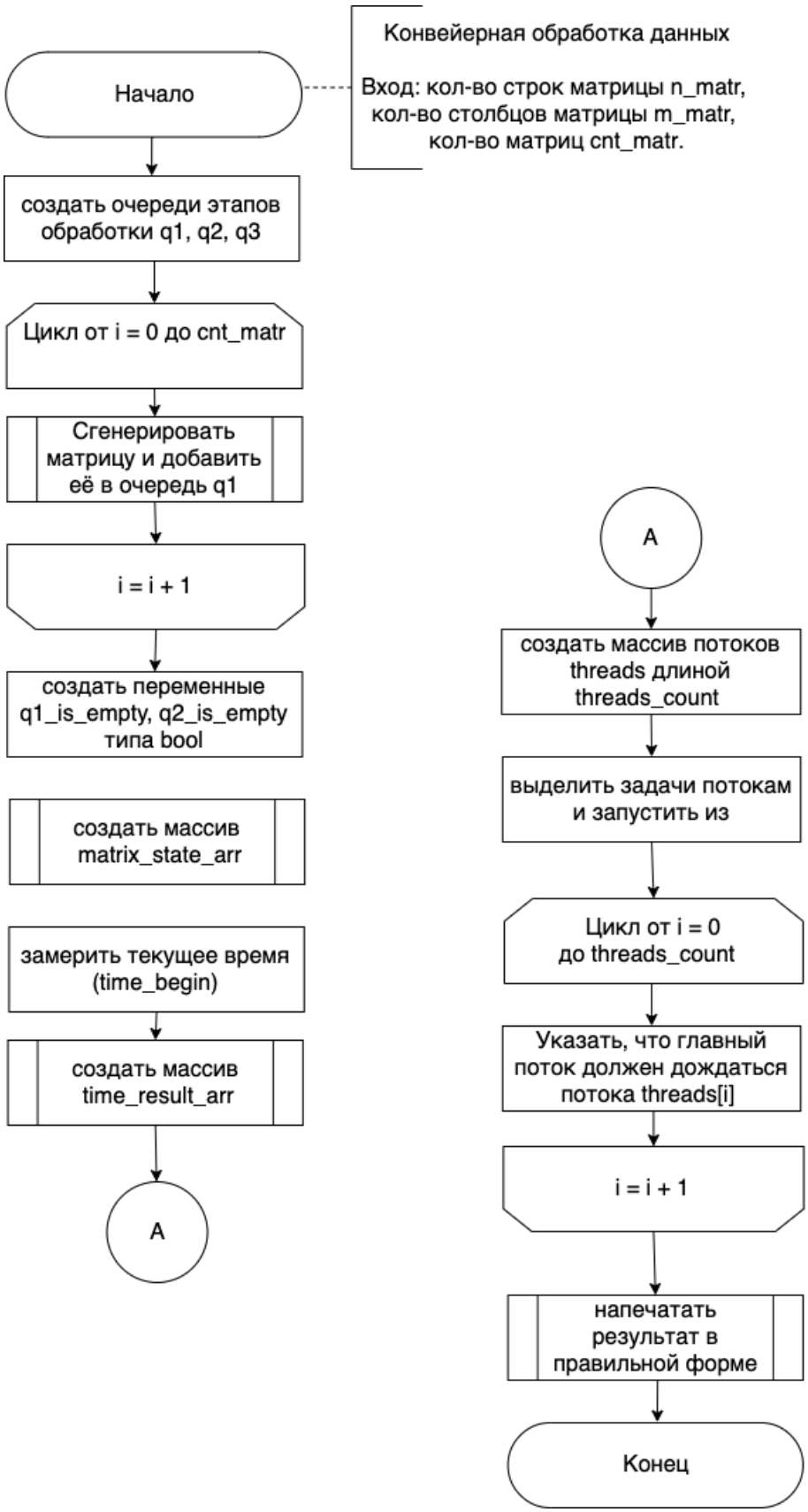

Рисунок 2.2 – Схема алгоритма конвейерной обработки матроицы

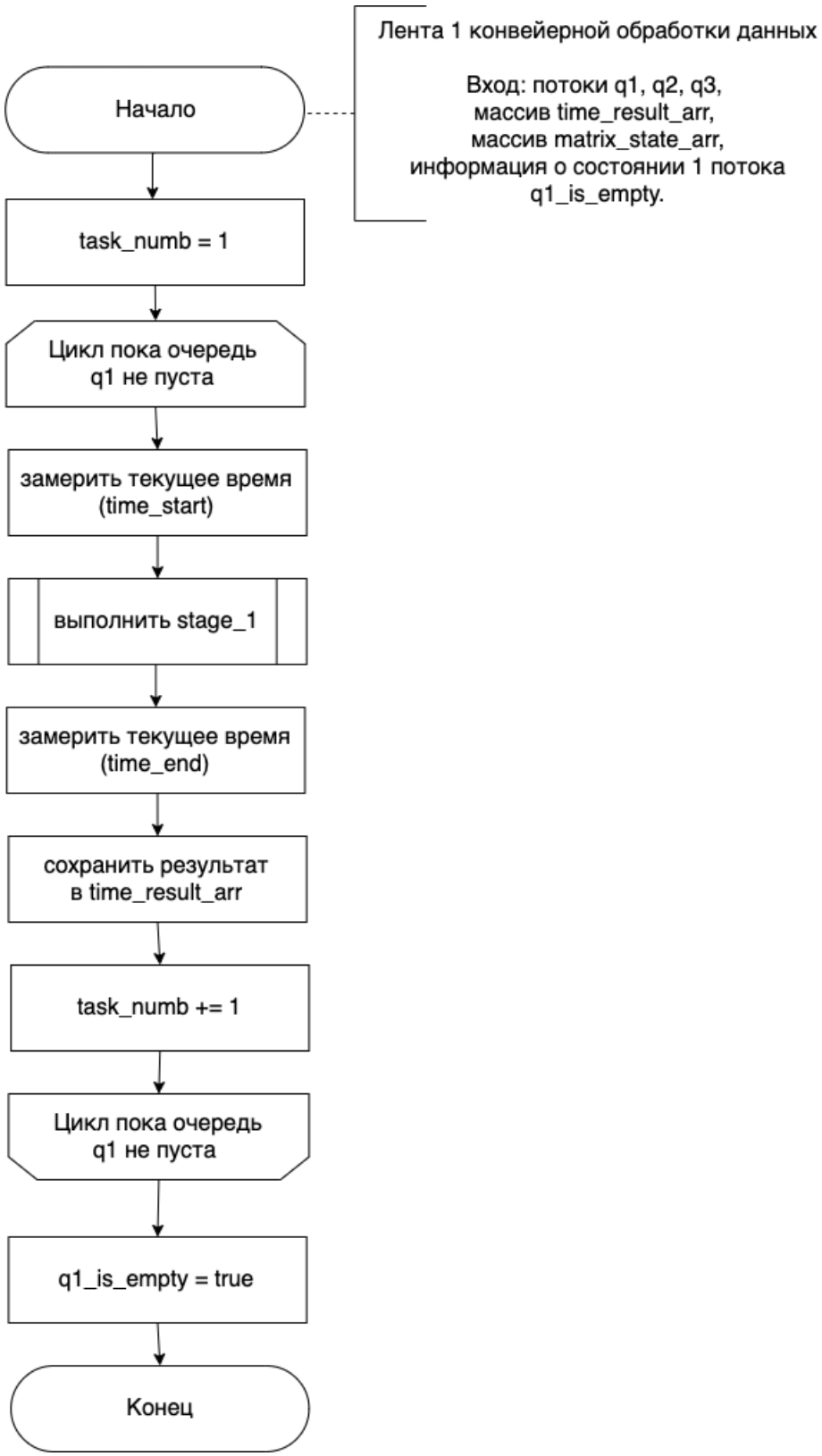

Рисунок 2.3 – Схема 1-ой ленты конвейерной обработки матрицы

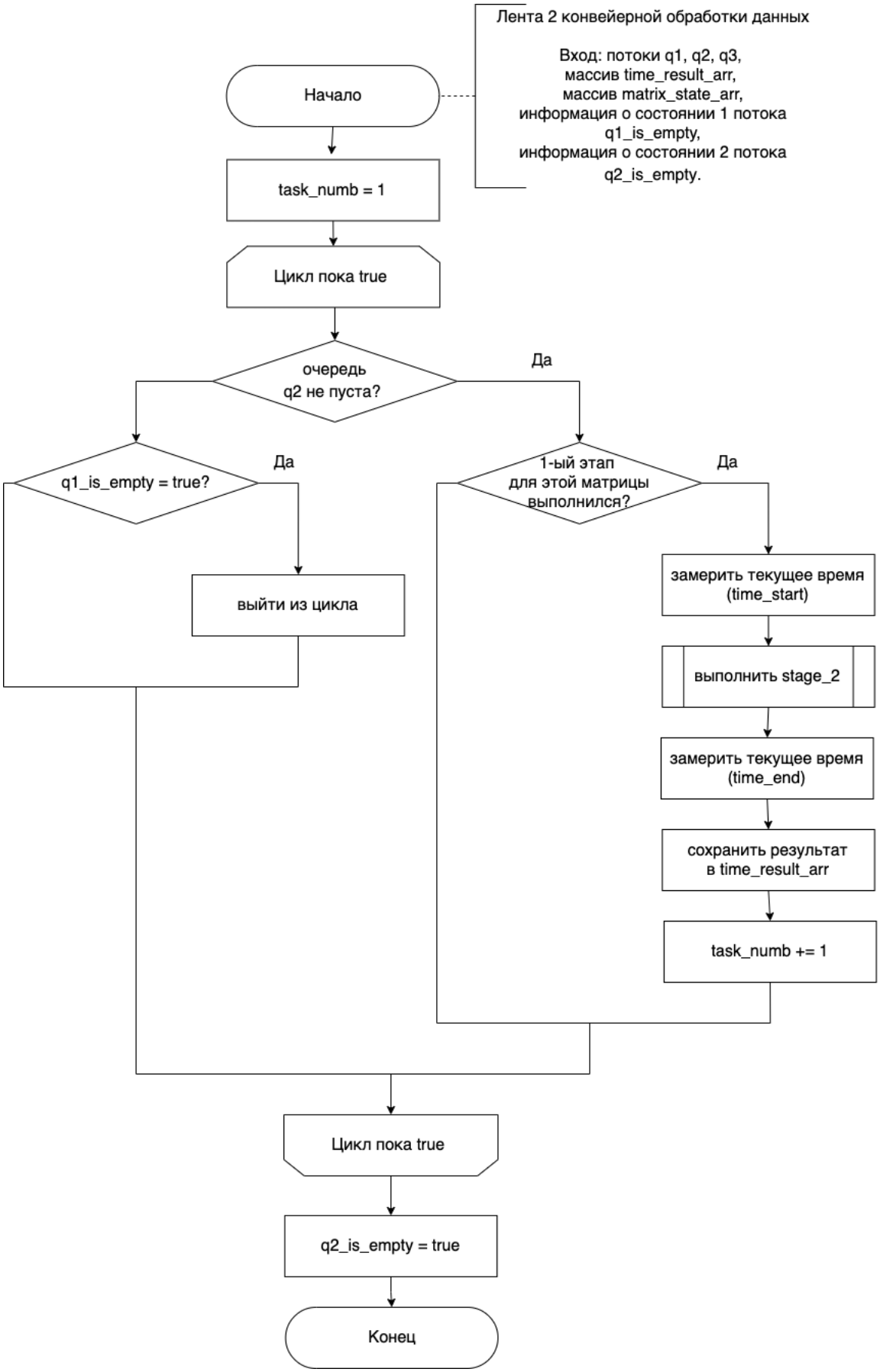

Рисунок 2.4 – Схема 2-ой ленты конвейерной обработки матрицы

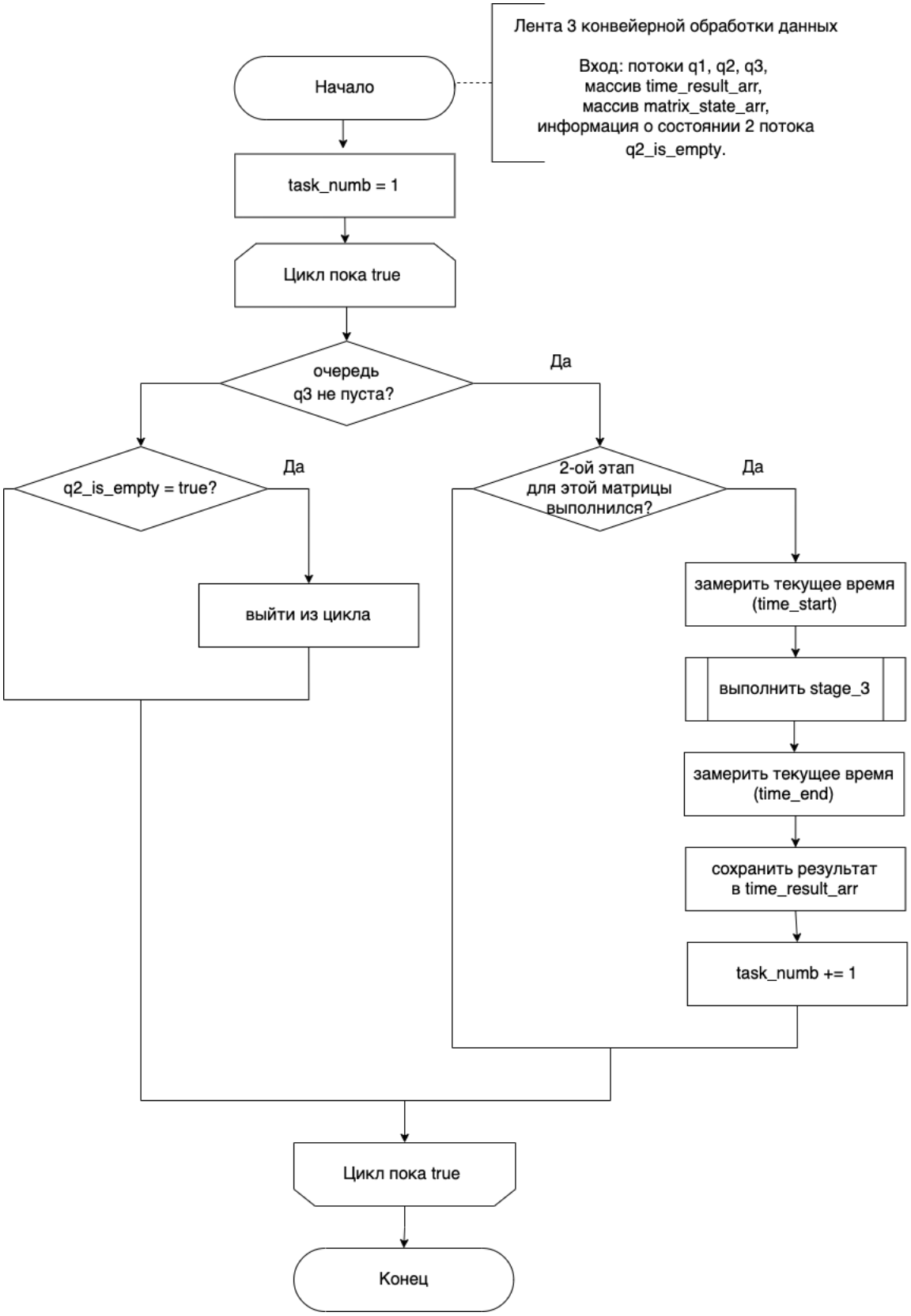

Рисунок 2.5 – Схема 3-ей ленты конвейерной обработки матрицы

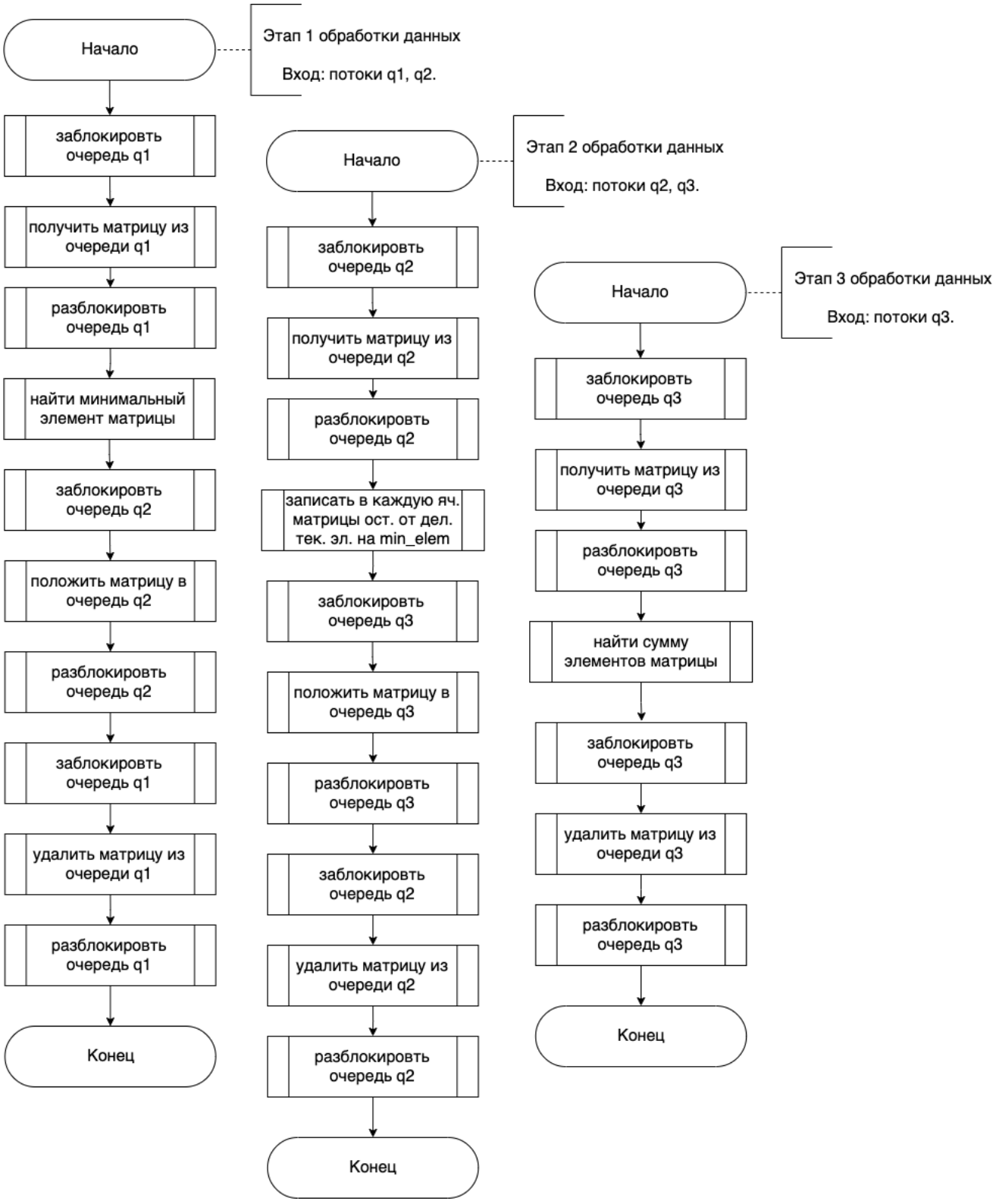

Рисунок 2.6 – Схема реализаций этапов обработки матроицы

2.2 Классы эквивалентности

Выделенные классы эквивалентности для тестирования:

- кол-во строк матрицы $<= 0$;
- кол-во столбцов матрицы $<= 0$;
- кол-во строк матрицы не является целым числом;
- кол-во столбцов матрицы не является целым числом;
- кол-во обрабатываемых матриц $<= 0$;
- кол-во обрабатываемых матриц не является целым числом;
- номер команды < 0 или > 3;
- номер команды не является целым числом;
- корректный ввод всех параметров;

2.3 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие структуры данных:

- кол-во матриц - целое число типа int;
- матрица - структура типа $matrix\_t$, имеющая следующие поля:
  - $std :: vector < std :: vector < int >>$ matr (сама матрица);
  - int n (кол-во строк в матрице);
  - int m (кол-во столбцов в матрице);
  - int min_elem (минимальный элемент в изначальной матрице);
  - int sum_elem (сумма элементов в конечной матрице);
- очереди имеют тип $std :: queue < matrix\_t >$;

- threads - массив потоков типа $std :: thread$ размера 3;
- структура состояния матрицы $matrix\_state$, которая содержит информацию, какие из этапов обработки матрицы выполнились, имеет следующие поля:
  - *bool* stage_1 (*true*, если 1-ый этап выполнился);
  - *bool* stage_2 (*true*, если 2-ой этап выполнился);
  - *bool* stage_3 (*true*, если 3-ий этап выполнился);
- время выполнения этапов обработки каждой матрицы записывается в массив типа res_time_t (*struct* result_time), эта структура имеет следующие поля:
  - *int* task (номер матрицы на ленте);
  - *int* tape (номер ленты);
  - *double* beg (время начала обработки этой матрицы на этой ленте);
  - *double* end (время конца обработки этой матрицы на этой ленте);
  - $std :: chrono :: time\_point < std :: chrono :: system\_clock >$ time_begin (время начала обработки матриц);

2.4 Структура ПО

ПО будет состоять из следующих модулей:

- *main.cpp* - файл, содержащий функцию *main*;
- *matrix.cpp* - файл, содержащий функции для работы с матрицей;
- *compare.cpp* - файл, в котором содержатся функции для замера времени работы алгоритмов;
- *read.cpp* - файл, в котором содержатся функции ввода данных;
- *conveyor.cpp* - файл, в котором содержатся функции для конвейерной и линейной обработок матриц;
- *errors.h* - файл, в котором содержатся классы для всех ошибок, которые могут возникнуть во время работы программы;
- *color.h* - файл, который содержит макросы для цветного вывода результата работы программы в консоль;
- *graph.py* - файл, содержащий функции для построения графиков зависимости времени от размера и кол-ва матриц;

Каждый файл с расширением *.cpp* имеет вспомогательный файл с расширением *.hpp*, где содержатся объявления функций, структуры, инклуды и дефайны.

2.5 Вывод

В данном разделе на основе теоретических данных были построены схемы требуемых методов обработки матриц (конвейерного и линейного), выбраны используемые типы данных, выделены классы эквивалентности для тестирования, а также была описана структура ПО.

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации, листинги кода, а также функциональные тесты.

3.1 Требования к программному обеспечению

- входные данные - кол-во строк и столбцов матрицы $matr$ должно быть > 0, все элементы матрицы имеют тип int, кол-во матриц > 0;
- выходные данные - табличка 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 данной работе для реализации был выбран язык программирования *C++* [2]. Выбор обсуловлен наличием опыта работы с ним. Время работы было замерено с помощью функции *std::chrono::system_clock::now()* [3].

3.3 Листинги кода

В листингах 3.1 - 3.8 представлены функции для конвейерного и ленейного алгоритмов обработки матриц.

Листинг 3.1 – Функция алгоритма 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;
    std::queue<matrix_t> q2;
    std::queue<matrix_t> 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("Первоначальная матрица:\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),
        std::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);
    }
}
```

Листинг 3.2 – Функция алгоритма 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;
    std::queue<matrix_t> q2;
    std::queue<matrix_t> 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("Первоначальная матрица:\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);
    }
}
```

Листинг 3.3 – Функция 1-ой ленты конвейерной обработки матрицы

```
void parallel_stage_1(std::queue<matrix_t> &q1, std::queue<matrix_t> &q2,
                      std::vector<res_time_t> &time_result_arr,
                      std::vector<matrix_state_t> &matrix_state_arr,
                      bool &q1_is_empty)
{
    std::chrono::time_point<std::chrono::system_clock> time_start, time_end;
    int task_num = 1;

    while(!q1.empty())
    {
        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_result_arr[0].time_begin, task_num, 1);

        matrix_state_arr[task_num - 1].stage_1 = true;
        task_num++;
    }

    q1_is_empty = true;
}
```

Листинг 3.4 – Функция 2-ой ленты конвейерной обработки матрицы

```
void parallel_stage_2(std::queue<matrix_t> &q2, std::queue<matrix_t> &q3,
                      std::vector<res_time_t> &time_result_arr,
                      std::vector<matrix_state_t> &matrix_state_arr,
                      bool &q1_is_empty, bool &q2_is_empty)
{
    std::chrono::time_point<std::chrono::system_clock> time_start, time_end;
    int task_num = 1;

    while(true)
    {
        if (!q2.empty())
        {
            if (matrix_state_arr[task_num - 1].stage_1 == true)
            {
                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_result_arr[0].time_begin, task_num, 2);

                matrix_state_arr[task_num - 1].stage_2 = true;
                task_num++;
            }
        }
        else if(q1_is_empty)
        {
            break;
        }
    }

    q2_is_empty = true;
}
```

Листинг 3.5 – Функция 3-ей ленты конвейерной обработки матрицы

```
void parallel_stage_3(std::queue<matrix_t> &q3,
                      std::vector<res_time_t> &time_result_arr,
                      std::vector<matrix_state_t> &matrix_state_arr,
                      bool &q2_is_empty, int cnt_matr, bool matr_is_print)
{
    std::chrono::time_point<std::chrono::system_clock> time_start, time_end;
    int task_num = 1;

    while(true)
    {
        if (!q3.empty())
        {
            if (matrix_state_arr[task_num - 1].stage_2 == true)
            {
                time_start = std::chrono::system_clock::now();
                stage_3(std::ref(q3), task_num, cnt_matr, matr_is_print);
                time_end = std::chrono::system_clock::now();

                save_result(time_result_arr, time_start, time_end,
                   time_result_arr[0].time_begin, task_num, 3);

                matrix_state_arr[task_num - 1].stage_3 = true;
                task_num++;
            }
        }
        else if(q2_is_empty)
        {
            break;
        }
    }
}
```

Листинг 3.6 – Функция реализации 1-ого этапа 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();
}
```

Листинг 3.7 – Функция реализации 2-ого этапа 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();
}
```

Листинг 3.8 – Функция реализации 3-его этапа 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\nМатрица после 2 этапа:\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

3.5 Вывод

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

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование представлены далее.

- Операционная система: macOS 11.5.2. [4]
- Память: 8 GiB.
- Процессор: 2,3 GHz 4-ядерный процессор Intel Core i5. [5]

При тестировании ноутбук был включен в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также системой тестирования.

4.2 Демонстрация работы программы

```
                Меню

        1. Линейная обработка
        2. Конвейерная обработка
        3. Замеры время
        0. Выход

        Выбор:
2

Введите кол-во строк матрицы: 4000
Введите кол-во столбцов матрицы: 4000

Введите кол-во матриц: 8

-------------------------------------------------------------
  № Матрицы  |    № Этапа   | Время начала | Время конца
-------------------------------------------------------------
      1      |      1       |   0.000128   |   0.430541
      1      |      2       |   0.430543   |   0.937640
      1      |      3       |   0.937643   |   1.364411
-------------------------------------------------------------
      2      |      1       |   0.430542   |   0.907557
      2      |      2       |   0.937642   |   1.426259
      2      |      3       |   1.426261   |   1.882938
-------------------------------------------------------------
      3      |      1       |   0.907559   |   1.344999
      3      |      2       |   1.426260   |   1.935759
      3      |      3       |   1.935762   |   2.362281
-------------------------------------------------------------
      4      |      1       |   1.345002   |   1.772208
      4      |      2       |   1.935762   |   2.409714
      4      |      3       |   2.409716   |   2.825306
-------------------------------------------------------------
      5      |      1       |   1.772211   |   2.240448
      5      |      2       |   2.409715   |   2.862726
      5      |      3       |   2.862729   |   3.248657
-------------------------------------------------------------
      6      |      1       |   2.240450   |   2.657119
      6      |      2       |   2.862729   |   3.298454
      6      |      3       |   3.298456   |   3.710215
-------------------------------------------------------------
      7      |      1       |   2.657121   |   3.055923
      7      |      2       |   3.298455   |   3.761232
      7      |      3       |   3.761234   |   4.157704
-------------------------------------------------------------
      8      |      1       |   3.055925   |   3.457034
      8      |      2       |   3.761233   |   4.210561
      8      |      3       |   4.210563   |   4.622953
-------------------------------------------------------------
```

Рисунок 4.1 – Пример работы программы (конвейерная реализация)

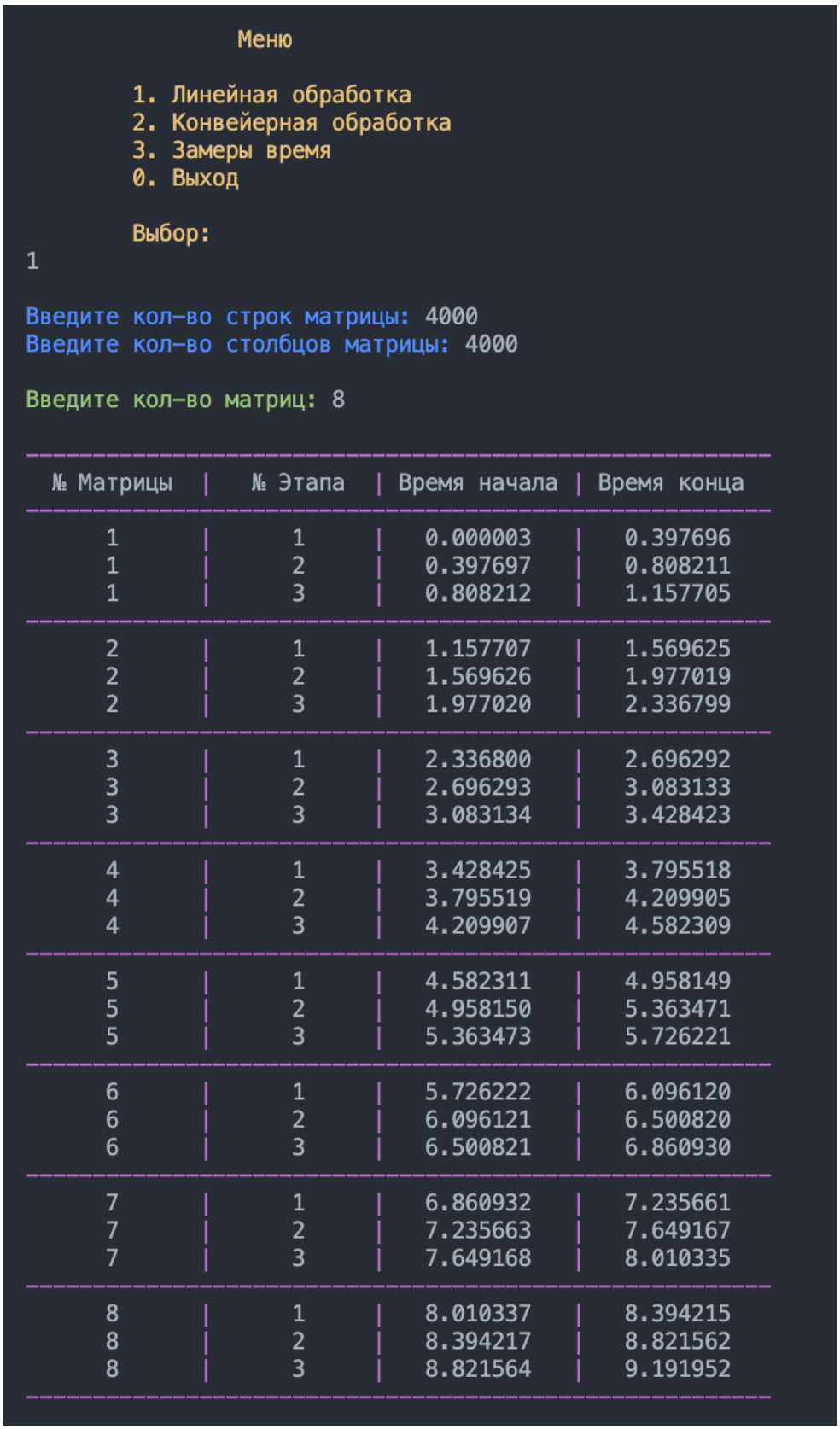

Рисунок 4.2 – Пример работы программы (линейная реализация)

4.3 Время выполнения алгоритмов

Результаты замеров времени работы алгоритмов обработки матриц для конвейерной и ленейной реализаций представлены на рисунках 4.3 - 4.5. Замеры времени проводились в секундах и усреднялись для каждого набора одинаковых экспериментов.

Зависимость времени от кол-ва матриц для Конвейерной обработки

Размер матриц	Кол-во матриц	Время конца
100	50	0.008400
100	100	0.012959
100	200	0.033233
100	400	0.042608
100	800	0.083202
100	1600	0.157260

Зависимость времени от кол-ва матриц для Линейной обработки

Размер матриц	Кол-во матриц	Время конца
100	50	0.015424
100	100	0.027467
100	200	0.054710
100	400	0.107318
100	800	0.216504
100	1600	0.421379

Зависимость времени от размера матриц для Конвейерной обработки

Размер матриц	Кол-во матриц	Время конца
20	100	0.001169
40	100	0.003011
80	100	0.007220
160	100	0.022288
320	100	0.081844
640	100	0.310014

Зависимость времени от размера матриц для Линейной обработки

Размер матриц	Кол-во матриц	Время конца
20	100	0.003016
40	100	0.007714
80	100	0.018300
160	100	0.061450
320	100	0.235350
640	100	0.899033

Рисунок 4.3 – Замеры времени работы алгоритмов для конвейерной и ленейной реализаций

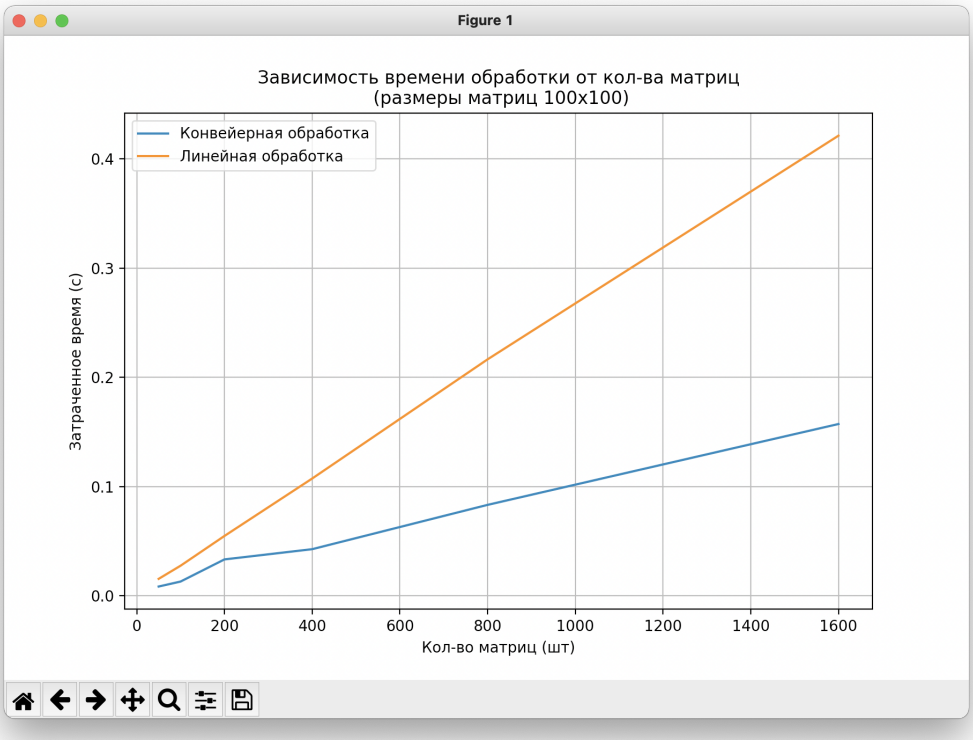

Рисунок 4.4 – Зависимость времени работы алгоритмов от кол-ва матриц (размеры матриц 100x100)

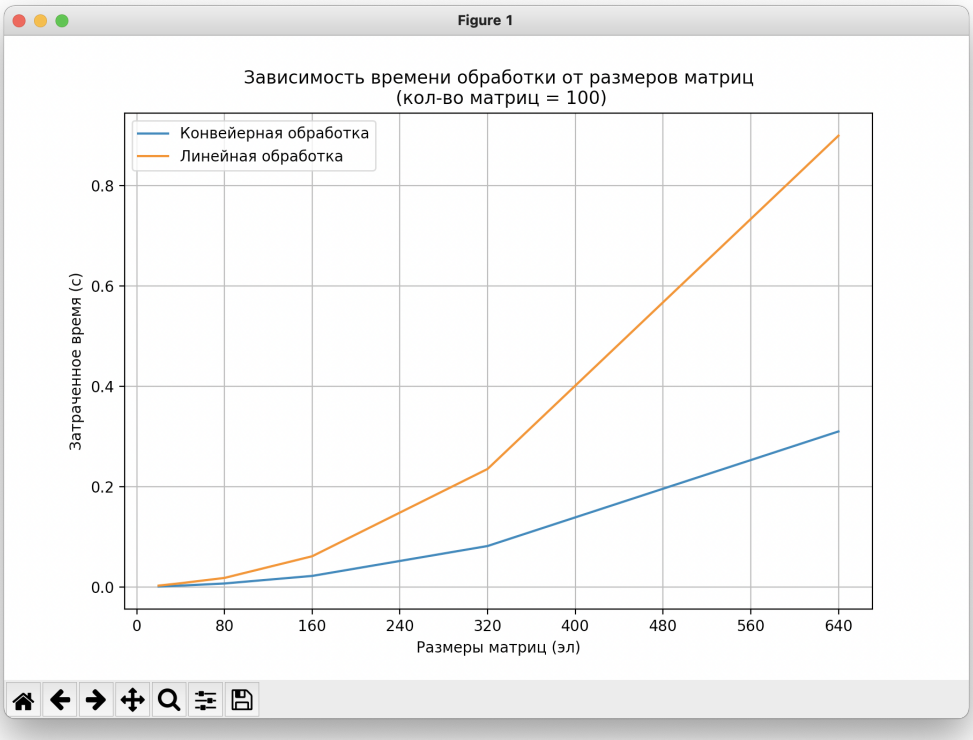

Рисунок 4.5 – Зависимость времени работы алгоритмов от размера матриц (кол-во матриц = 100)

4.4 Вывод

В этом разделе были указаны технические характеристики машины, на которой происходило сравнение времени работы алгоритмов обработки матриц для конвейерной и ленейной реализаций.

В результате замеров времени было установлено, что конвейерная реализация обработки лучше линейной при большом кол-ве 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 кол-ве обрабатываемых матриц, а так же при матрицах большого размера стоит использовать конвейерную реализацию обработки, а не линейную.

Заключение

Было экспериментально подтверждено различие во временной эффективности конвейрной и линейной реализаций обработак матриц. В результате исследований можно сделать вывод о том, что при большом кол-ве 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).

В ходе выполнения данной лабораторной работы были решены следующие задачи:

- изучены основы конвейерной обработки данных;
- применены изученные основы для реализации конвейерной обработки матриц;
- произведен сравнительный анализ линейной и конвейерной реализаций обработки матриц;
- экспериментально подтверждено различие во временной эффективности линейной и конвейерной реализаций обработки матриц при помощи разработанного программного обеспечения на материале замеров процессорного времени;
- описаны и обоснованы полученные результаты в отчете о выполненной лабораторной работе.

Поставленная цель была достигнута.

Литература

[1] Конвейерная организация [Электронный ресурс]. Режим доступа: http://www.citforum.mstu.edu.ru/hardware/svk/glava_5.shtml (дата обращения: 01.12.2021).

[2] C++ — Типизированный язык программирования / Хабр [Электронный ресурс]. Режим доступа: https://habr.com/ru/hub/cpp/ (дата обращения: 20.10.2021).

[3] std::chrono::system_clock::now - cppreference.com [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/cpp/chrono/system_clock/now (дата обращения: 20.10.2021).

[4] macOS Monterey - Apple(RU) [Электронный ресурс]. Режим доступа: https://www.apple.com/ru/macos/monterey/ (дата обращения: 14.10.2021).

[5] Intel [Электронный ресурс]. Режим доступа: https://www.intel.ru/content/www/ru/ru/products/details/processors/core/i5.html (дата обращения: 14.10.2021).