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 Косарев А.А.

Группа ИУ7-51Б

Оценка (баллы)

Преподаватель Волкова Л. Л., Строганов Ю.В.

Москва — 2023 г.

Содержание

Введение **3**

1 Аналитическая часть **4**

1.1 Описание конвейерной обработки данных 4

1.2 Описание алгоритмов 4

2 Конструкторская часть **6**

2.1 Алгоритмы обработки матриц 6

2.2 Классы эквивалентности 13

3 Технологическая часть **14**

3.1 Требования к программному обеспечению 14

3.2 Выбор языка программирования 14

3.3 Реализация алгоритмов 15

3.4 Функциональные тесты 24

4 Исследовательская часть **25**

4.1 Технические характеристики устройства 25

4.2 Время выполнения алгоритмов 26

4.3 Вывод 27

Заключение **28**

Список использованных источников **29**

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;

— 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.

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. Реализуемая программа 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.

2.1 Алгоритмы обработки матриц

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.

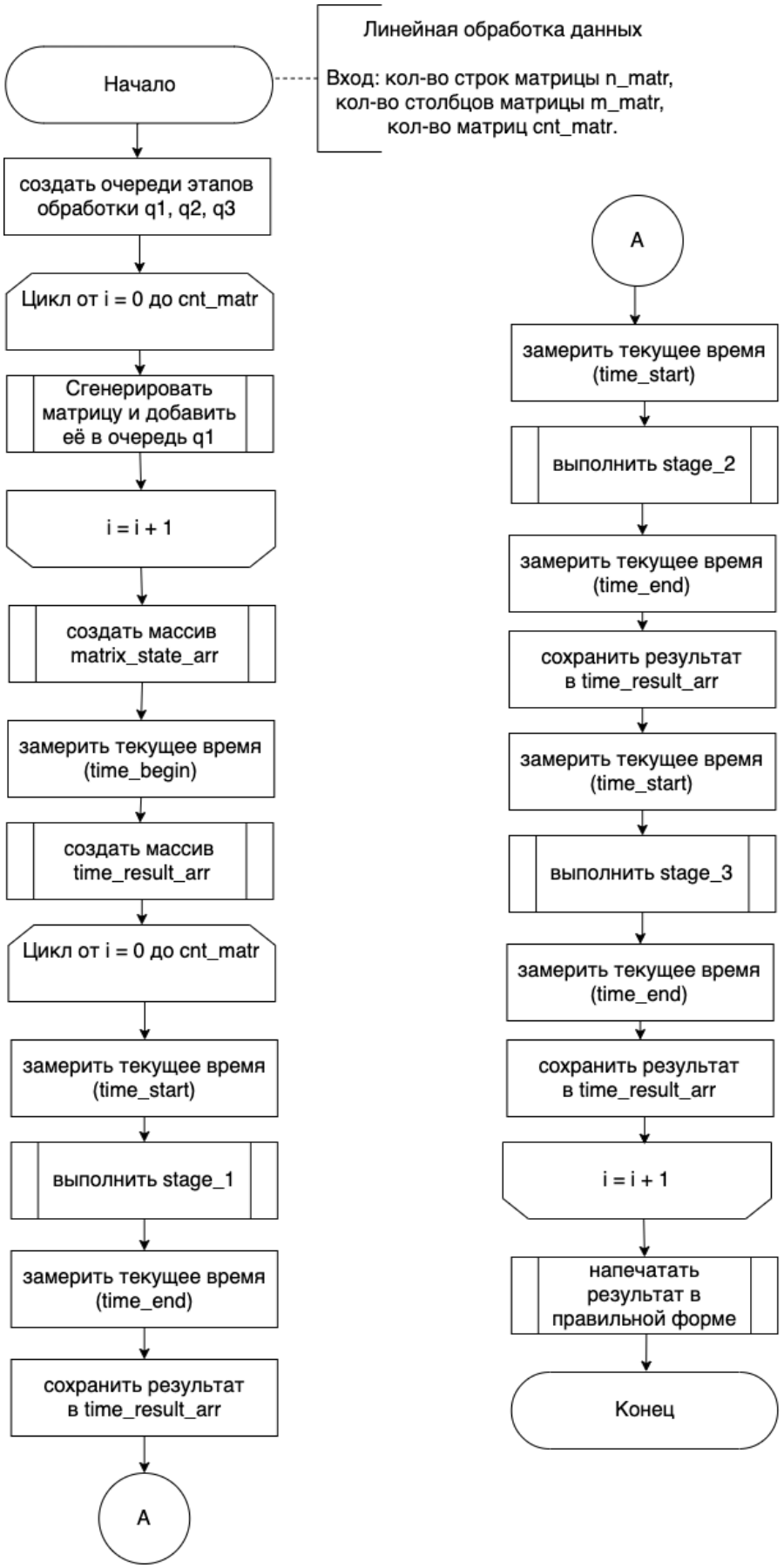

Рисунок 2.1 – Схема алгоритма линейной обработки матрицы

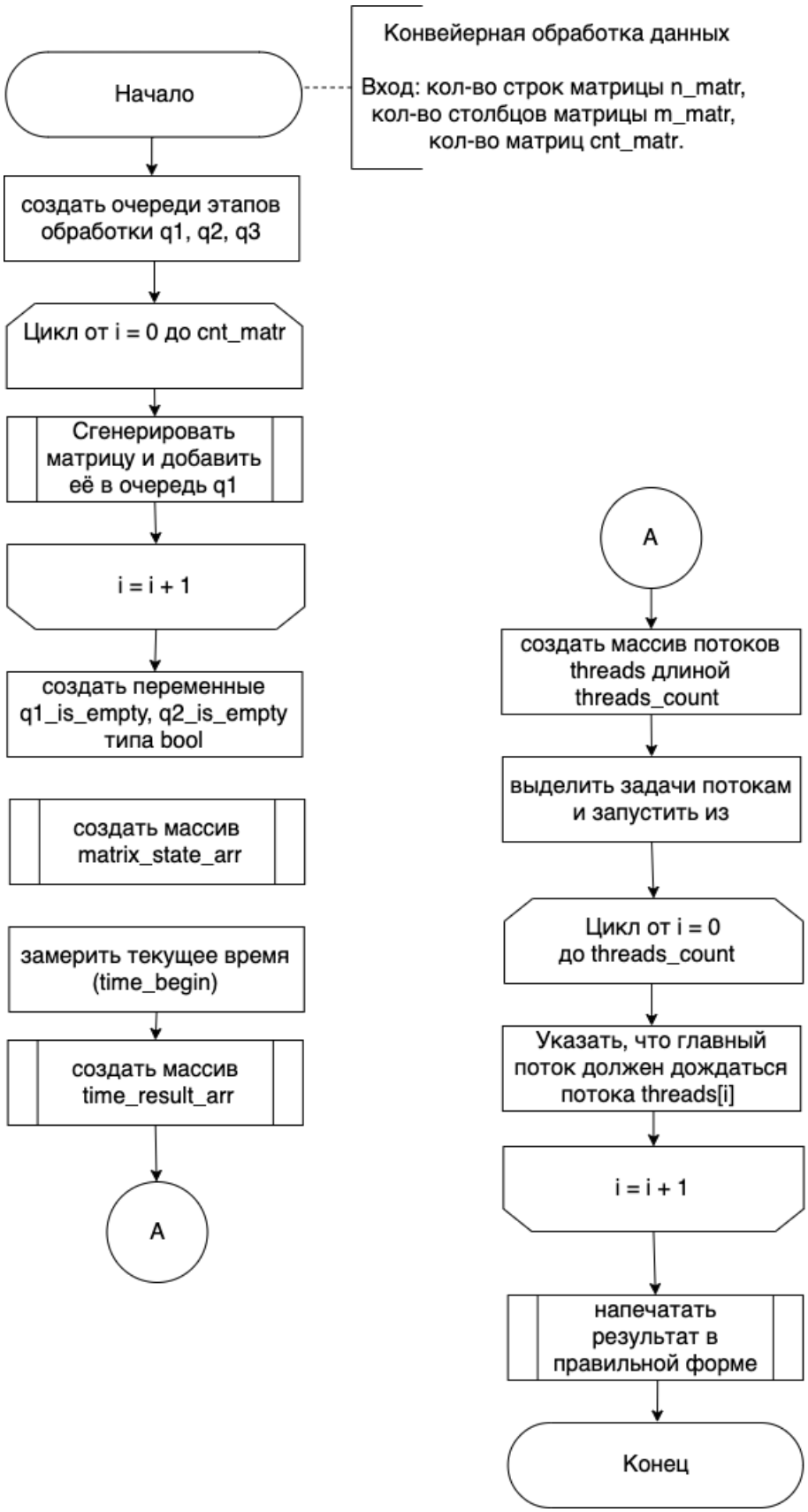

Рисунок 2.2 – Схема алгоритма конвейерной обработки матрицы

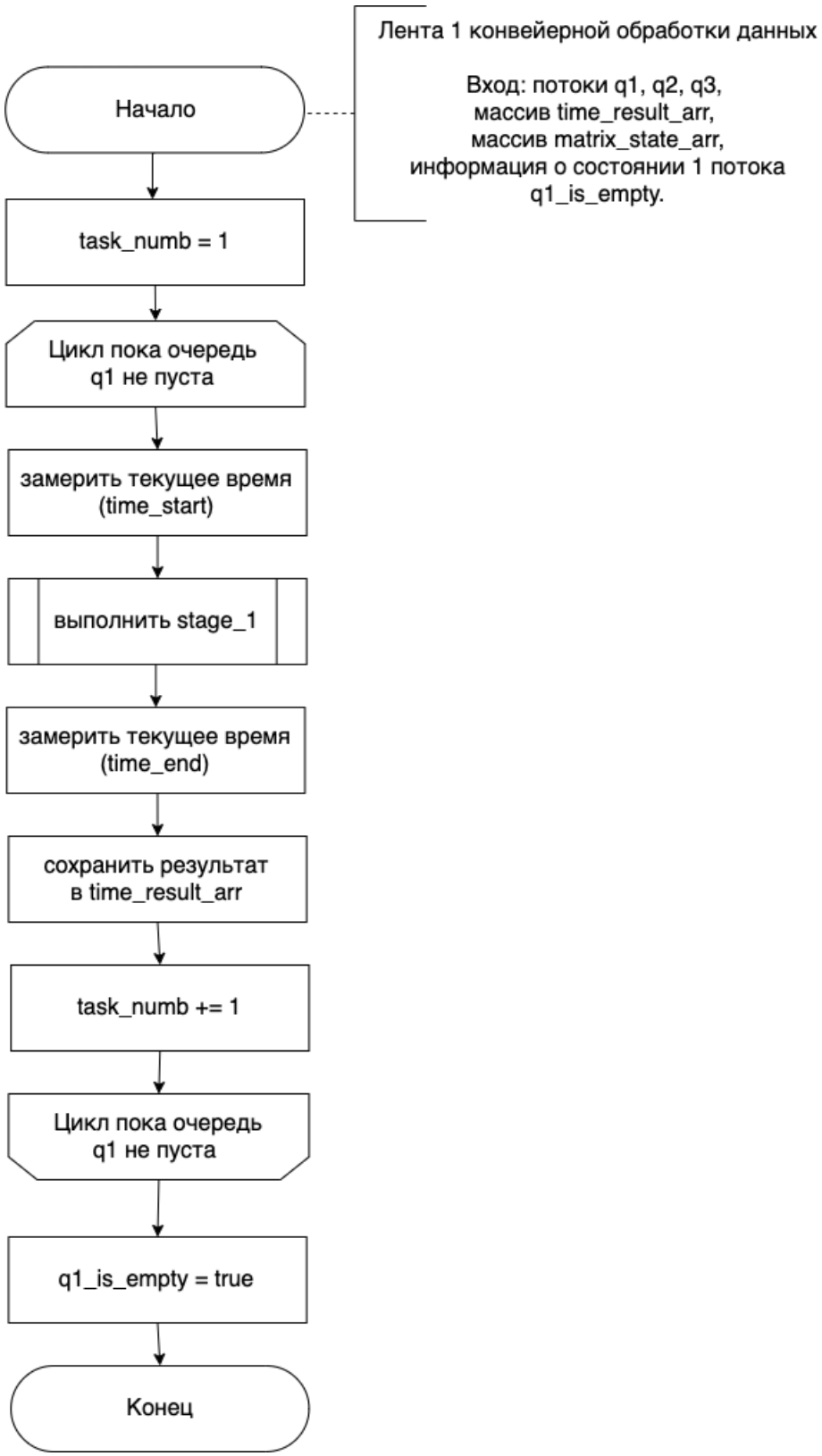

Рисунок 2.3 – Схема 1-ой ленты конвейерной обработки матрицы

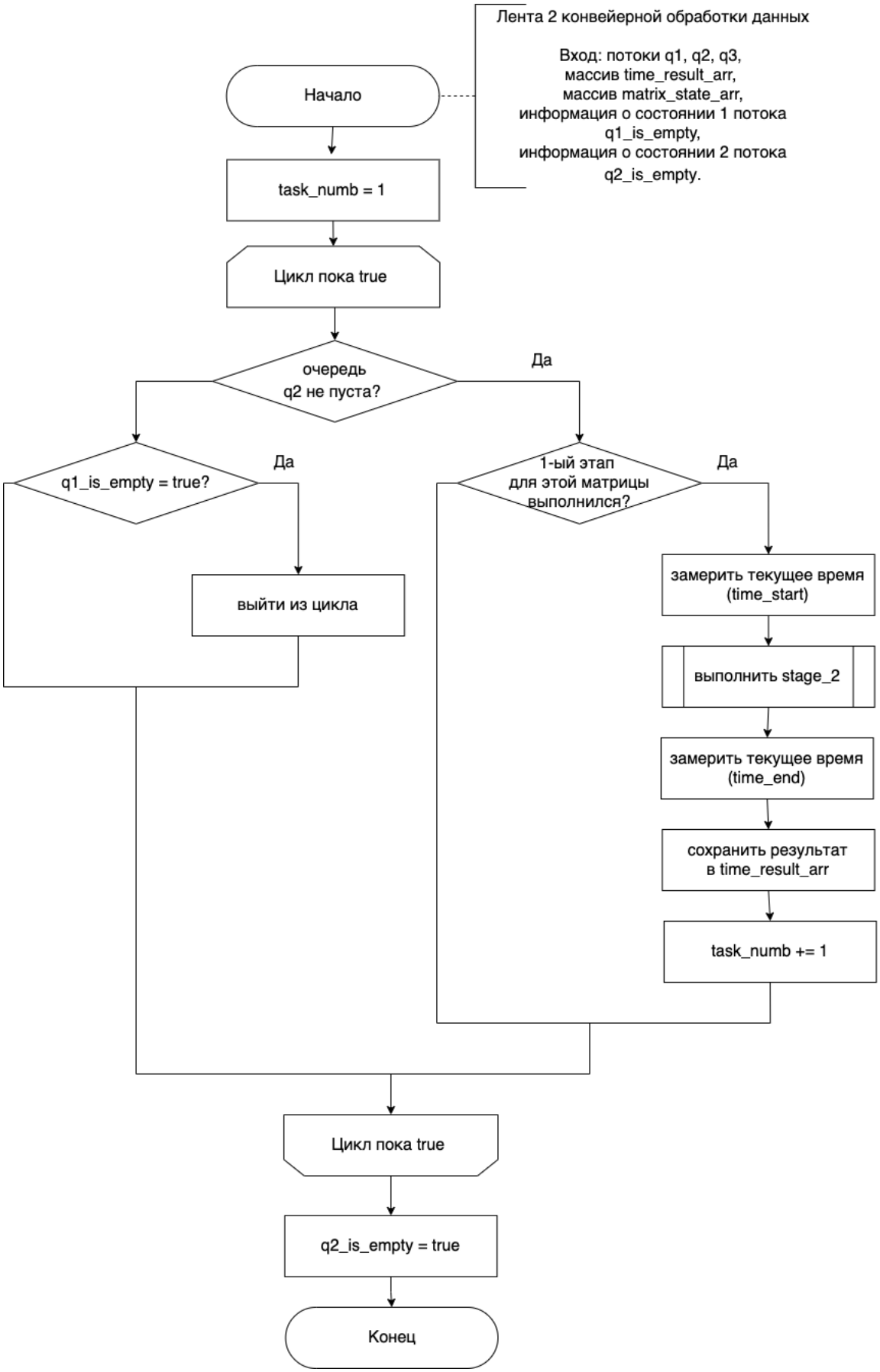

Рисунок 2.4 – Схема 2-ой ленты конвейерной обработки матрицы

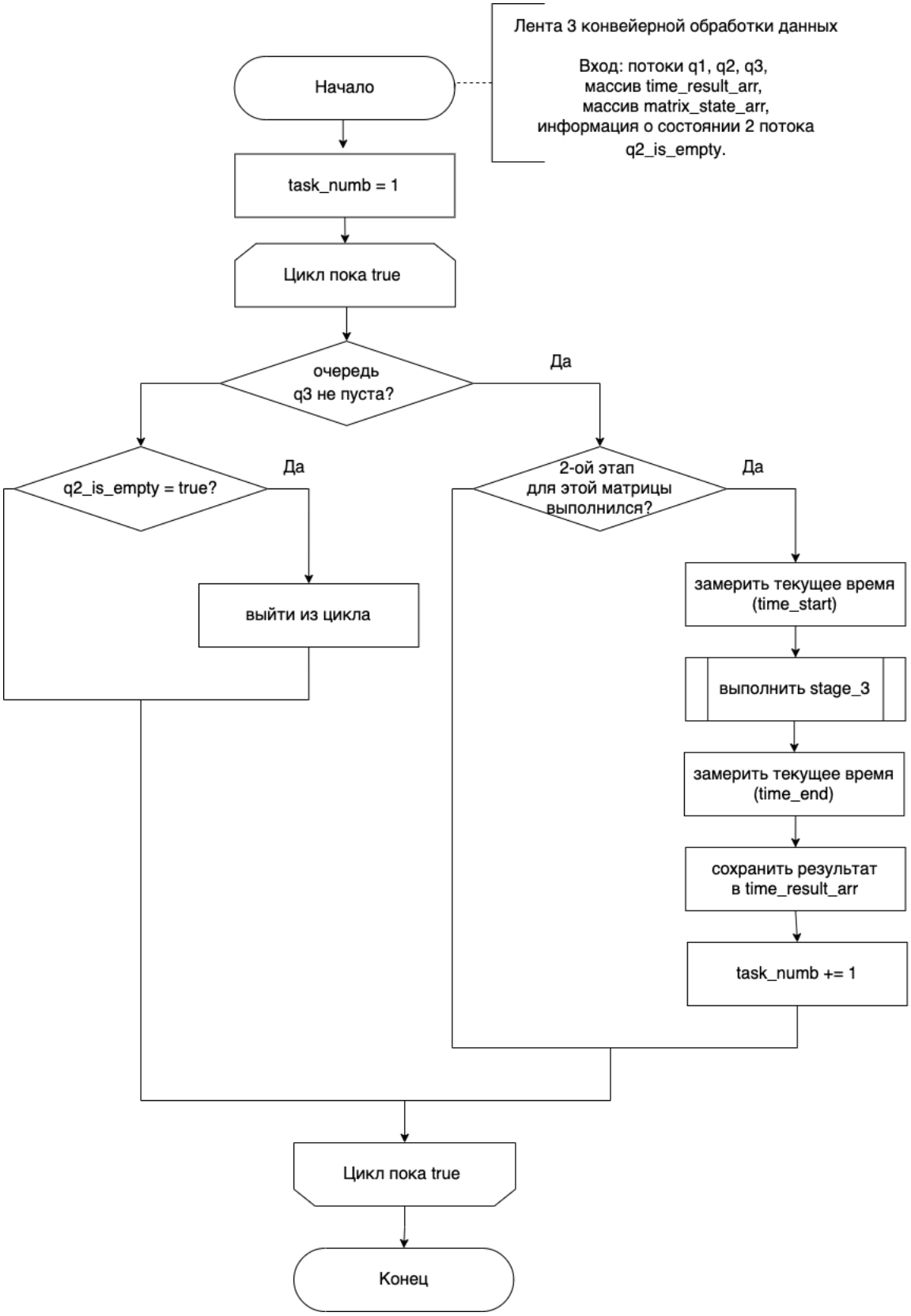

Рисунок 2.5 – Схема 3-ей ленты конвейерной обработки матрицы

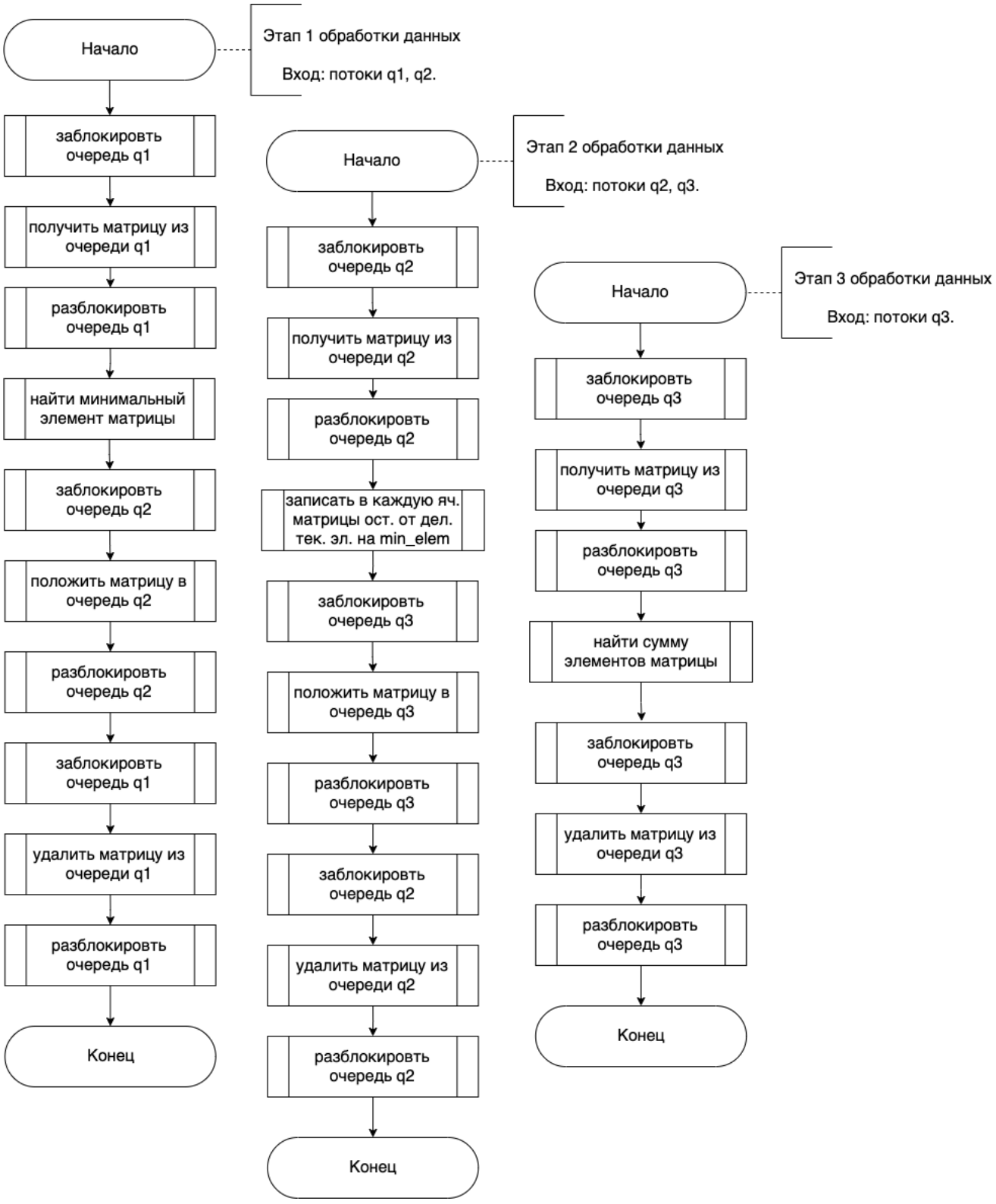

Рисунок 2.6 – Схема реализаций этапов обработки матрицы

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;

Вывод

В данном разделе на основе теоретических данных были построены схемы требуемых методов обработки матриц (конвейерного и линейного), выделены классы эквивалентности для тестирования.

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

В качестве входных данных задается количество строк и столбцов матрицы $matr$, которое должно быть больше 0, а все элементы матрицы имеют тип int. Количество матриц больше 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 Выбор языка программирования

В данной работе для реализации был выбран язык программирования *C++* [2], так как он предоставляет весь необходимый функционал для выполнения работы. Для замера времени работы использовалась функция *std::chrono::system_clock::now()* [3].

3.3 Реализация алгоритмов

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

Листинг 3.1 – Функция алгоритма конвейерной обработки матрицы

```
1  void parallel_processing(int n_matr, int m_matr, int cnt_matr, bool
       matr_is_print, bool compare_time)
2  {
3      std::queue<matrix_t> q1;
4      std::queue<matrix_t> q2;
5      std::queue<matrix_t> q3;
6
7      std::mutex m;
8
9      for (int i = 0; i < cnt_matr; i++)
10     {
11         matrix_t matr = generate_matrix(n_matr, m_matr);
12         q1.push(matr);
13
14         if (matr_is_print && i == cnt_matr - 1)
15         {
16             m.lock();
17             printf("                                    ␣
                                  :\n");
18             print_matrix(matr);
19             m.unlock();
20         }
21     }
22
23     bool q1_is_empty = false;
24     bool q2_is_empty = false;
25
26     std::vector<matrix_state_t> matrix_state_arr;
27     init_matrix_state_arr(matrix_state_arr, cnt_matr);
28
29     std::chrono::time_point<std::chrono::system_clock> time_begin =
30     std::chrono::system_clock::now();
31
32     std::vector<res_time_t> time_result_arr;
```

```
33      init_time_result_arr(time_result_arr, time_begin, cnt_matr,
            THREADS_COUNT);
34
35      std::thread threads[THREADS_COUNT];
36
37      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));
38      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));
39      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);
40
41      for (int i = 0; i < THREADS_COUNT; i++)
42      {
43          threads[i].join();
44      }
45
46      if (compare_time)
47      {
48          printf("     %4d       %s|%s     %4d       %s|%s    %.6f  \n",
49              n_matr, PURPLE, BASE_COLOR,
50              cnt_matr, PURPLE, BASE_COLOR,
51              time_result_arr[cnt_matr - 1].end);
52      }
53      else
54      {
55          print_res_time(time_result_arr, cnt_matr * THREADS_COUNT);
56      }
57  }
```

Листинг 3.2 – Функция алгоритма линейной обработки матрицы

```
1  void linear_processing(int n_matr, int m_matr, int cnt_matr, bool
       matr_is_print, bool compare_time)
2  {
3      std::queue<matrix_t> q1;
4      std::queue<matrix_t> q2;
5      std::queue<matrix_t> q3;
6
7      std::mutex m;
8
9      for (int i = 0; i < cnt_matr; i++)
10     {
11         matrix_t matr = generate_matrix(n_matr, m_matr);
12         q1.push(matr);
13
14         if (matr_is_print && i == cnt_matr - 1)
15         {
16             m.lock();
17             printf("                                    ␣
                                  :\n");
18             print_matrix(matr);
19             m.unlock();
20         }
21     }
22
23     std::vector<matrix_state_t> matrix_state_arr;
24     init_matrix_state_arr(matrix_state_arr, cnt_matr);
25
26     std::chrono::time_point<std::chrono::system_clock> time_start,
           time_end,
27     time_begin = std::chrono::system_clock::now();
28
29     std::vector<res_time_t> time_result_arr;
30     init_time_result_arr(time_result_arr, time_begin, cnt_matr,
           THREADS_COUNT);
31
32     for (int i = 0; i < cnt_matr; i++)
33     {
34         time_start = std::chrono::system_clock::now();
35         stage_1(std::ref(q1), std::ref(q2));
36         time_end = std::chrono::system_clock::now();
```

```
37
38          save_result(time_result_arr, time_start, time_end,
                time_begin, i + 1, 1);
39
40          time_start = std::chrono::system_clock::now();
41          stage_2(std::ref(q2), std::ref(q3));
42          time_end = std::chrono::system_clock::now();
43
44          save_result(time_result_arr, time_start, time_end,
                time_begin, i + 1, 2);
45
46          time_start = std::chrono::system_clock::now();
47          stage_3(std::ref(q3), i + 1, cnt_matr, matr_is_print);
48          time_end = std::chrono::system_clock::now();
49
50          save_result(time_result_arr, time_start, time_end,
                time_begin, i + 1, 3);
51      }
52
53      if (compare_time)
54      {
55          printf("␣␣␣␣␣%4d␣␣␣␣␣␣%s|%s␣␣␣␣␣%4d␣␣␣␣␣␣%s|%s␣␣␣%.6f␣␣\n",
56              n_matr, PURPLE, BASE_COLOR,
57              cnt_matr, PURPLE, BASE_COLOR,
58              time_result_arr[cnt_matr - 1].end);
59      }
60      else
61      {
62          print_res_time(time_result_arr, cnt_matr * THREADS_COUNT);
63      }
64  }
```

Листинг 3.3 – Функция 1-ой ленты конвейерной обработки матрицы

```
1  void parallel_stage_1(std::queue<matrix_t> &q1,
       std::queue<matrix_t> &q2,
2                        std::vector<res_time_t> &time_result_arr,
3                        std::vector<matrix_state_t> &matrix_state_arr,
4                        bool &q1_is_empty)
5  {
6      std::chrono::time_point<std::chrono::system_clock> time_start,
           time_end;
```

```
7       int task_num = 1;
8
9       while(!q1.empty())
10      {
11          time_start = std::chrono::system_clock::now();
12          stage_1(std::ref(q1), std::ref(q2));
13          time_end = std::chrono::system_clock::now();
14
15          save_result(time_result_arr, time_start, time_end,
                time_result_arr[0].time_begin, task_num, 1);
16
17          matrix_state_arr[task_num - 1].stage_1 = true;
18          task_num++;
19      }
20
21      q1_is_empty = true;
22  }
```

Листинг 3.4 – Функция 2-ой ленты конвейерной обработки матрицы

```
1  void parallel_stage_2(std::queue<matrix_t> &q2,
       std::queue<matrix_t> &q3,
2                        std::vector<res_time_t> &time_result_arr,
3                        std::vector<matrix_state_t> &matrix_state_arr,
4                        bool &q1_is_empty, bool &q2_is_empty)
5  {
6      std::chrono::time_point<std::chrono::system_clock> time_start,
           time_end;
7      int task_num = 1;
8
9      while(true)
10     {
11         if (!q2.empty())
12         {
13             if (matrix_state_arr[task_num - 1].stage_1 == true)
14             {
15                 time_start = std::chrono::system_clock::now();
16                 stage_2(std::ref(q2), std::ref(q3));
17                 time_end = std::chrono::system_clock::now();
18
19                 save_result(time_result_arr, time_start, time_end,
                       time_result_arr[0].time_begin, task_num, 2);
20
21                 matrix_state_arr[task_num - 1].stage_2 = true;
22                 task_num++;
23             }
24         }
25         else if(q1_is_empty)
26         {
27             break;
28         }
29     }
30
31     q2_is_empty = true;
32 }
```

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

```
1  void stage_3(std::queue<matrix_t> &q3, int task_num, int cnt_matr,
       bool matr_is_print)
2  {
3      std::mutex m;
4
5      m.lock();
6      matrix_t matr = q3.front();
7      m.unlock();
8
9      matr.sum_elem = get_sum_elements(matr);
10
11     if (matr_is_print && task_num == cnt_matr)
12     {
13         printf("\nmin_elem␣=␣%d\n\ n                ␣          ␣2␣
                       :\n", matr.min_elem);
14         print_matrix(matr);
15         printf("\nsum_elem␣=␣%d\n\n", matr.sum_elem);
16     }
17
18     m.lock();
19     q3.pop();
20     m.unlock();
21 }
```

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 ленейного алгоритмов обработки матриц, проведено тестирование, описаны средства реализации и требования к программе.

4 Исследовательская часть

4.1 Технические характеристики устройства

Ниже представлены характеристики компьютера, на котором проводилось тестирование программы:

— операционная система Windows 10 Домашняя 21H2;

— оперативная память 16 Гб;

— процессор Intel(R) Core(TM) i7-10870H CPU @ 2.20 ГГц.

Во время тестирования ноутбук был подключен к сети электропитания. Из программного обеспечения были запущены только среда разработки *Clion* и браузер *Chrome*.

Процессор был загружен на 19%, оперативная память – на 50%.

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.

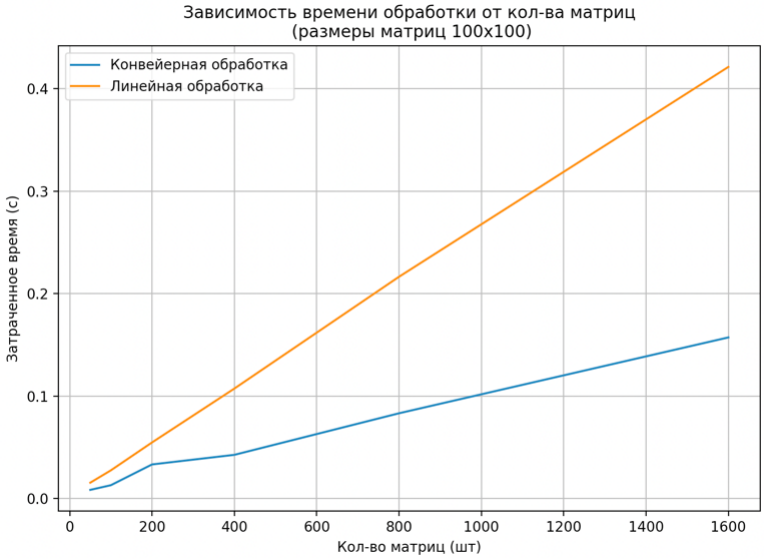

Рисунок 4.1 – Зависимость времени работы алгоритмов от кол-ва матриц (размеры матриц 100x100)

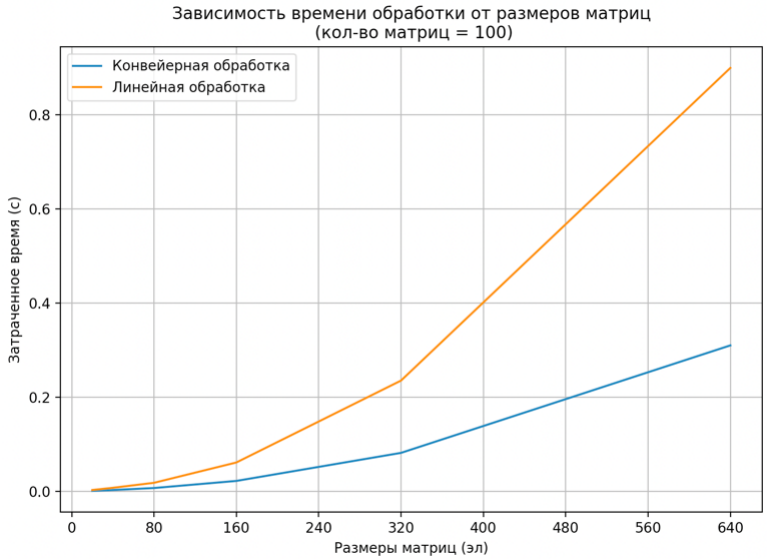

Рисунок 4.2 – Зависимость времени работы алгоритмов от размера матриц (кол-во матриц = 100)

4.3 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.

Список использованных источников

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

2. C++ — Типизированный язык программирования / Хабр [Электронный ресурс]. — URL: https://habr.com/ru/hub/cpp/ (дата обращения: 01.12.2021).

3. std::chrono::system_clock::now - cppreference.com [Электронный ресурс]. — URL: https://en.cppreference.com/w/cpp/chrono/system_clock/now (дата обращения: 05.12.2021).

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