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

Название: Конвейерная обработка данных

Дисциплина: Анализ алгоритмов

Студент ИУ7-54б _____ Ларин В.Н.
(группа) (подпись, дата) (фамилия, и.о.)

Преподаватель _____ Волкова Л.Л.
(подпись, дата) (фамилия, и.о.)

Москва, 2021

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 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;

— 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 Аналитический раздел

В этом разделе будет представлено описание сути конвейерной 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 конвейерной обработки данных будет обрабатываться матрица. В качестве алгоритмов на каждую из трех лент были выбраны следующие действия.

— Найти наименьший элемент в матрице.

— Записать в каждую ячейку матрицы остаток от деления текущего элемента на минимальный элемент, найденный в предыдущем конвейере.

— Найти сумму элементов полученной матрицы.

1.3 Требования к программному обеспечению

— входные данные - количество строк и столбцов матрицы должно быть > 0, все элементы матрицы имеют целый тип , количество матриц > 0;

— выходные данные - таблица с номерами матриц, номерами этапов (лент) её обработки, временем начала обработки текущей матрицы на текущей ленте, временем окончания обработки текущей матрицы на текущей ленте.

Реализуемое ПО должно давать возможность выбрать метод обработки данных (конвейерный или линейный) и вывести для него результат вычисления, а также возможность произвести сравнение алгоритмов по затраченному времени.

1.4 Вывод

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

2 Конструкторский раздел

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.

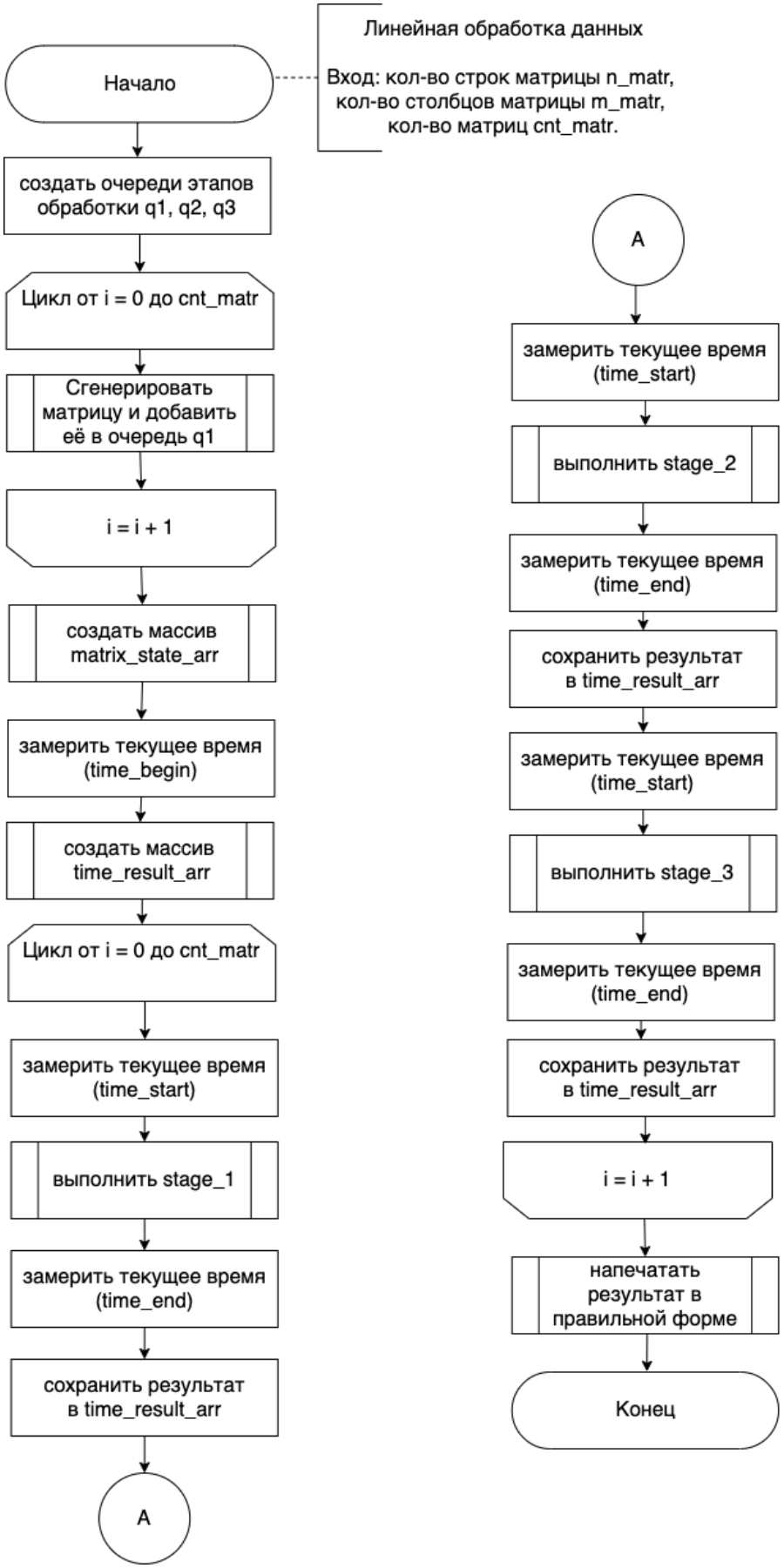

Рисунок 2.1 — Схема алгоритма линейной обработки матроицы

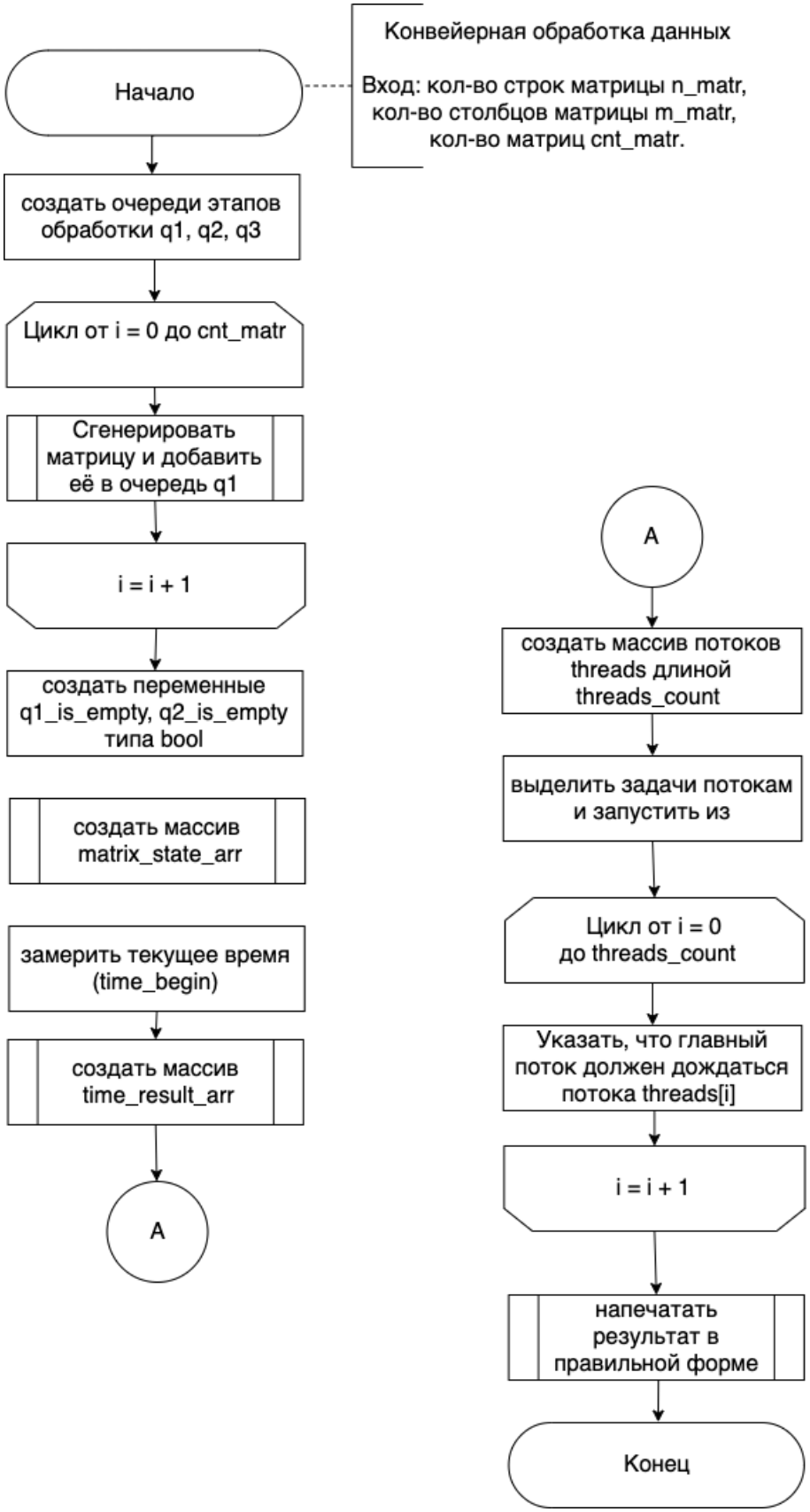

Рисунок 2.2 — Схема алгоритма конвейерной обработки матроицы

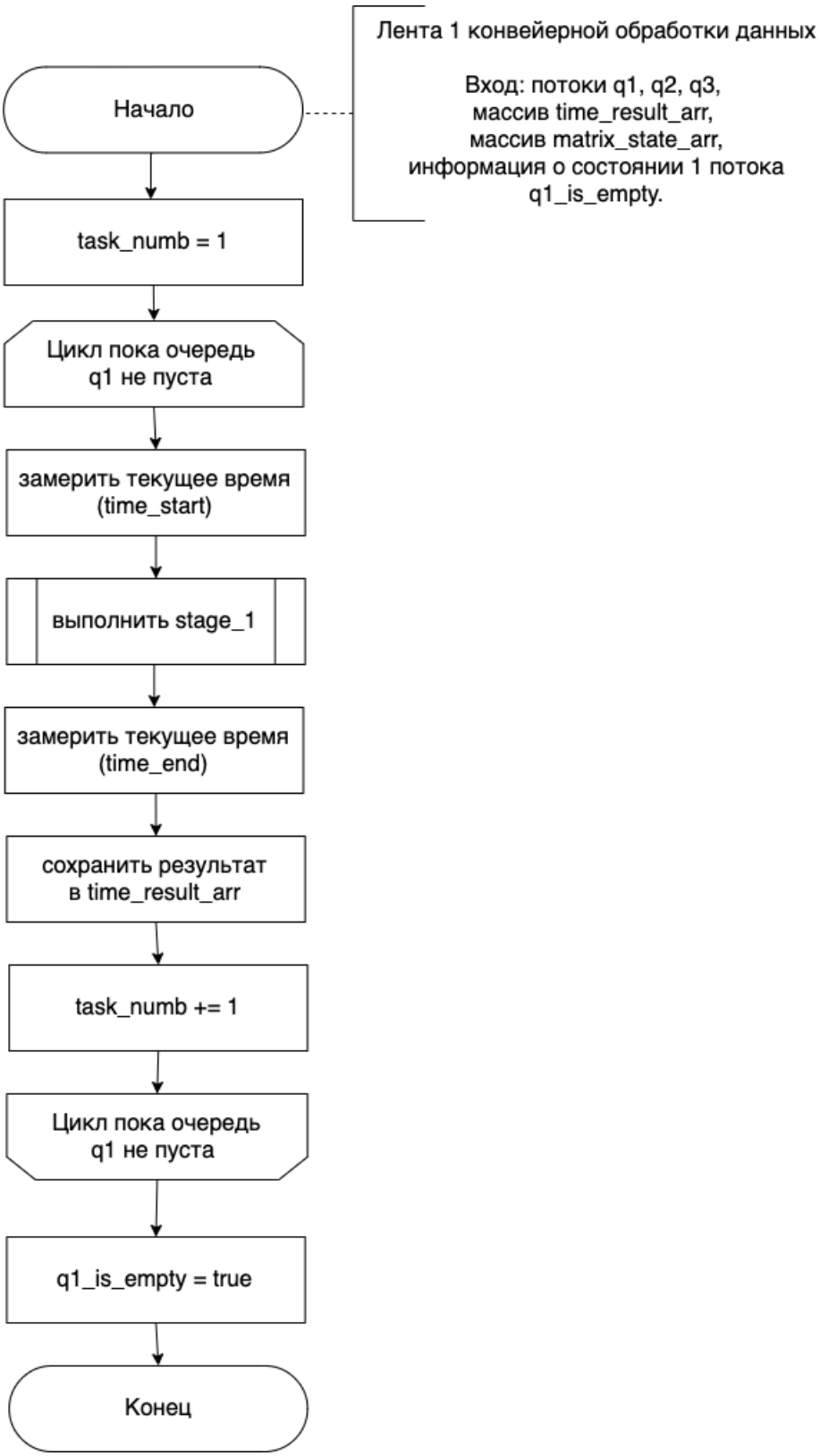

Рисунок 2.3 — Схема 1-ой ленты конвейерной обработки матрицы

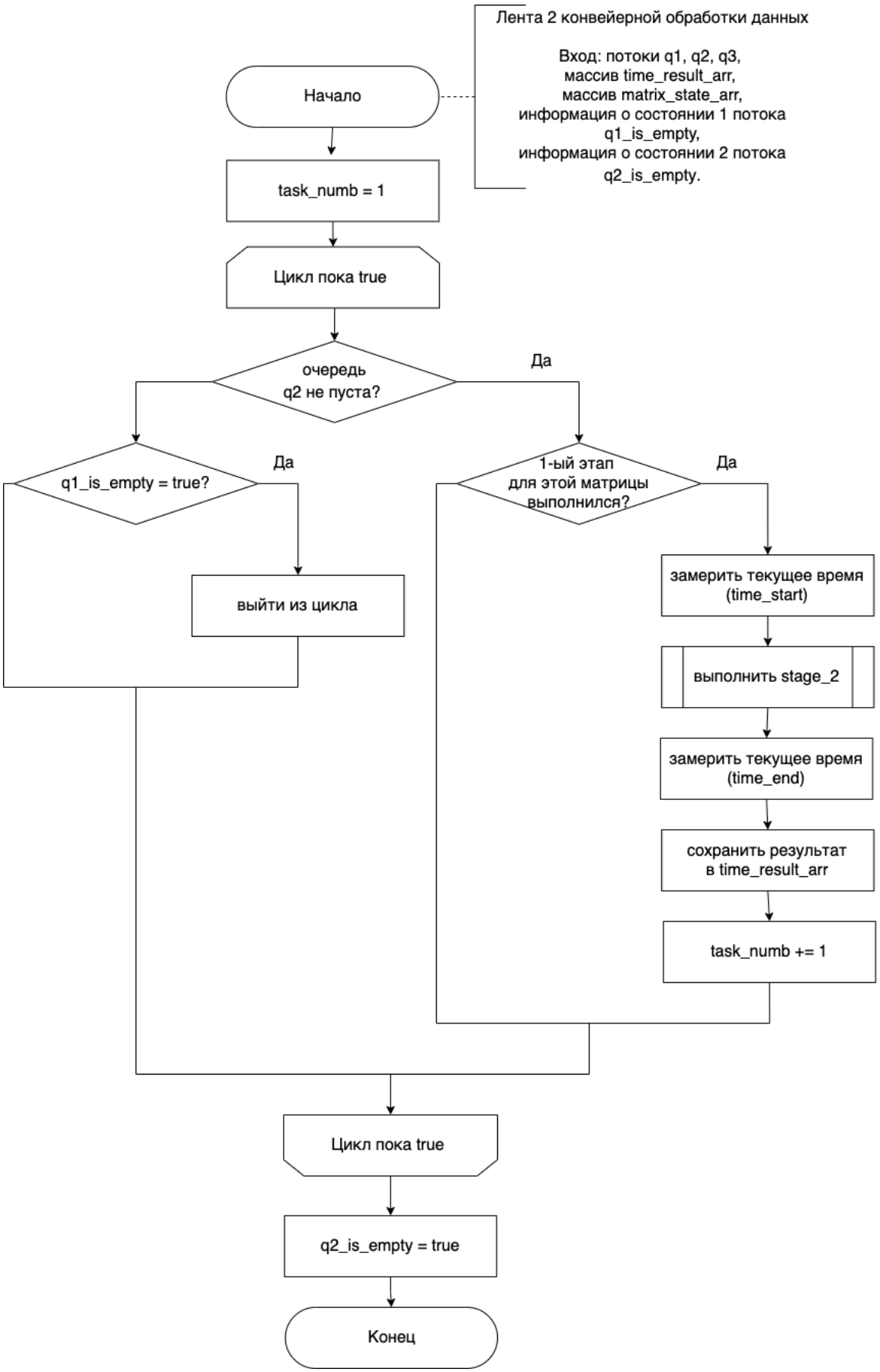

Рисунок 2.4 — Схема 2-ой ленты конвейерной обработки матрицы

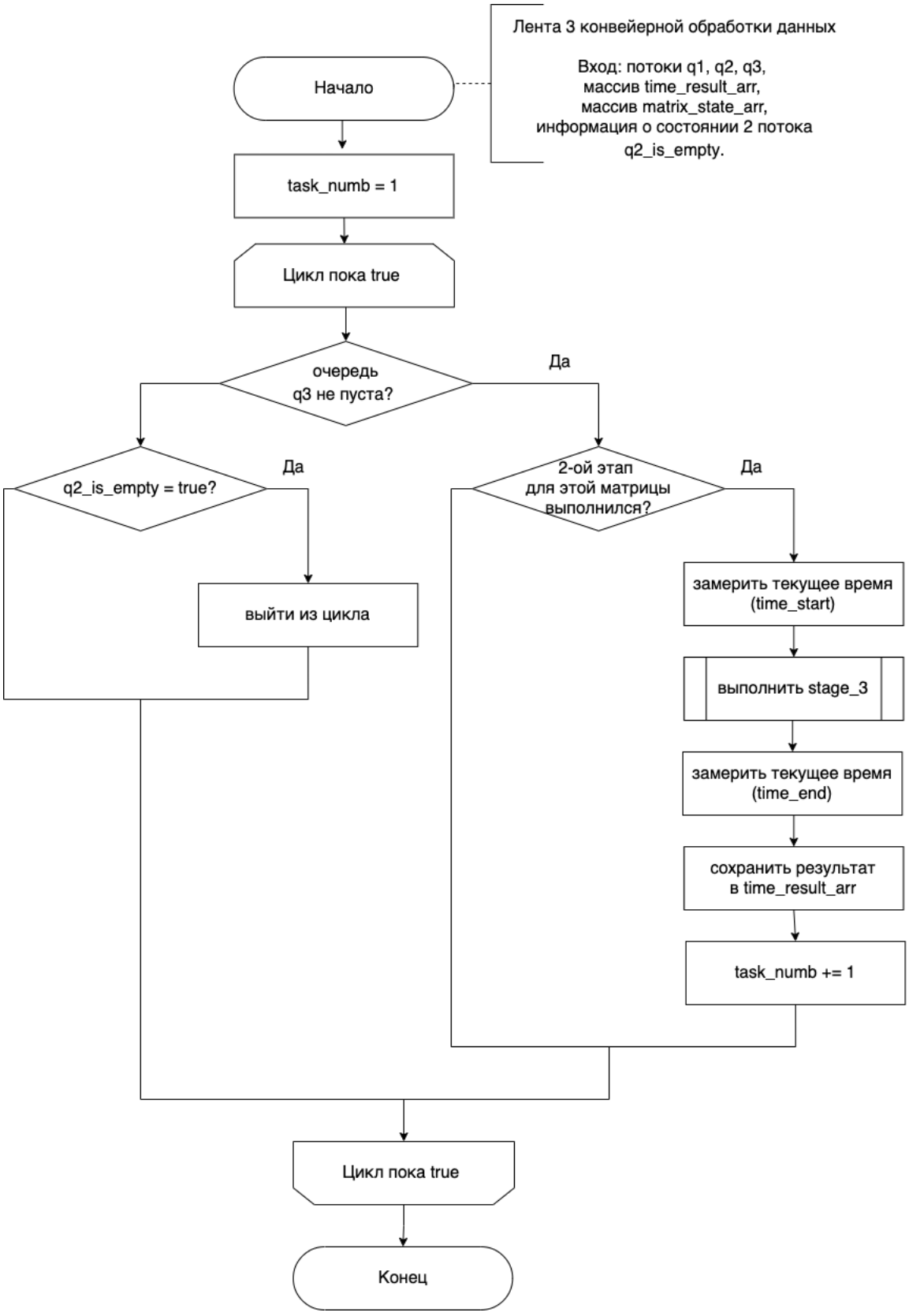

Рисунок 2.5 — Схема 3-ей ленты конвейерной обработки матрицы

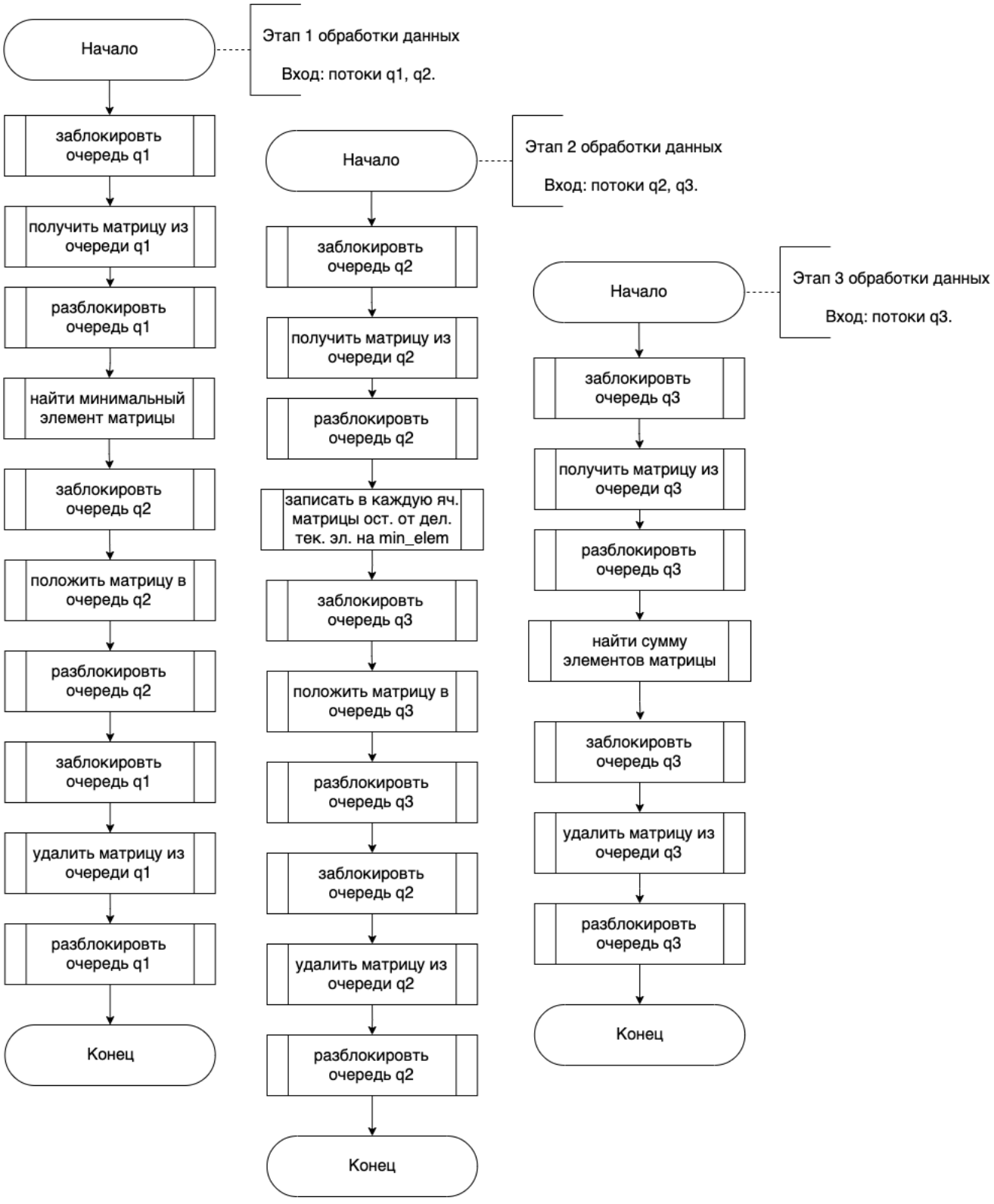

Рисунок 2.6 — Схема реализаций этапов обработки матроицы

2.2 Классы эквивалентности

Выделенные классы эквивалентности для тестирования:

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

2.3 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;

2.4 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 Технологический раздел

В данном разделе будут приведены средства реализации, листинги кода, а также функциональные тесты.

3.1 Средства реализации

Основным средством разработки является язык программирования. Был выбран язык программирования C++. Данный выбор обоснован высокой скоростью работы языка, поддержкой строгой типизации [2]. Для сборки проекта был выбран инструмент CMake [3]. В качестве среды разработки был выбран инструмент JetBrains Clion [4].

3.2 Листинги кода

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

Листинг 3.1 — Функция алгоритма конвейерной обработки матриц

```
1  void parallelProcessing(int rows, int cols, int count, bool print, bool
       compareTime) {
2      std::queue<Matrix> q1;
3      std::queue<Matrix> q2;
4      std::queue<Matrix> q3;
5
6      std::mutex m;
7
8      for (int i = 0; i < count; i++) {
9          Matrix matr(rows, cols, true);
10         q1.push(matr);
11     }
12
13     bool q1IsEmpty = false;
14     bool q2IsEmpty = false;
15
16     std::vector<MatrixState> matrixStateArr;
17     initMatrixStateArr(matrixStateArr, count);
18
19     std::chrono::time_point<std::chrono::system_clock> time_begin =
20             std::chrono::system_clock::now();
21
22     std::vector<ResultTime> timeResultArr;
23     initTimeResultArr(timeResultArr, time_begin, count, THREADS_COUNT);
24
25     std::thread threads[THREADS_COUNT];
26
```

```
27      threads[0] = std::thread(parallelStage1, std::ref(q1), std::ref(q2),
            std::ref(timeResultArr),
28                               std::ref(matrixStateArr),
                                     std::ref(q1IsEmpty));
29      threads[1] = std::thread(parallelStage2, std::ref(q2), std::ref(q3),
            std::ref(timeResultArr),
30                               std::ref(matrixStateArr),
                                     std::ref(q1IsEmpty), std::ref(q2IsEmpty));
31      threads[2] = std::thread(parallelStage3, std::ref(q3),
            std::ref(timeResultArr), std::ref(matrixStateArr),
32                               std::ref(q2IsEmpty), count, print);
33
34      for (auto & thread : threads) {
35          thread.join();
36      }
37  }
```

Листинг 3.2 — Функция алгоритма линейной обработки матрицы

```
1   void linearProcessing(int rows, int cols, int count, bool print, bool
        compareTime) {
2       std::queue<Matrix> q1;
3       std::queue<Matrix> q2;
4       std::queue<Matrix> q3;
5
6       std::mutex m;
7
8       for (int i = 0; i < count; i++) {
9           q1.push(Matrix(rows, cols, true));
10      }
11
12      std::vector<MatrixState> MatrixStateArr;
13      initMatrixStateArr(MatrixStateArr, count);
14
15      std::chrono::time_point<std::chrono::system_clock> timeStart, timeEnd,
16              time_begin = std::chrono::system_clock::now();
17
18      std::vector<ResultTime> time_result_arr;
19      initTimeResultArr(time_result_arr, time_begin, count, THREADS_COUNT);
20
21      for (int i = 0; i < count; i++) {
22          timeStart = std::chrono::system_clock::now();
23          stage1(std::ref(q1), std::ref(q2));
24          timeEnd = std::chrono::system_clock::now();
25
26          saveResult(time_result_arr, timeStart, timeEnd, time_begin, i + 1,
                1);
```

```
27
28          timeStart = std::chrono::system_clock::now();
29          stage2(std::ref(q2), std::ref(q3));
30          timeEnd = std::chrono::system_clock::now();
31
32          saveResult(time_result_arr, timeStart, timeEnd, time_begin, i + 1,
                2);
33
34          timeStart = std::chrono::system_clock::now();
35          stage3(std::ref(q3), i + 1, count, print);
36          timeEnd = std::chrono::system_clock::now();
37
38          saveResult(time_result_arr, timeStart, timeEnd, time_begin, i + 1,
                3);
39      }
40  }
```

Листинг 3.3 — Функция 1-ой ленты конвейерной обработки матрицы

```
1   void parallelStage1(std::queue<Matrix> &q1, std::queue<Matrix> &q2,
2                       std::vector<ResultTime> &time_result_arr,
3                       std::vector<MatrixState> &matrixStateArr,
4                       bool &q1IsEmpty) {
5       std::chrono::time_point<std::chrono::system_clock> timeStart, timeEnd;
6       int taskNum = 1;
7
8       while (!q1.empty()) {
9           timeStart = std::chrono::system_clock::now();
10          stage1(std::ref(q1), std::ref(q2));
11          timeEnd = std::chrono::system_clock::now();
12
13          saveResult(time_result_arr, timeStart, timeEnd,
                time_result_arr[0].timeBegin, taskNum, 1);
14
15          matrixStateArr[taskNum - 1].stage_1 = true;
16          taskNum++;
17      }
18
19      q1IsEmpty = true;
20  }
```

Листинг 3.4 — Функция 2-ой ленты конвейерной обработки матрицы

```
1   void parallelStage2(std::queue<Matrix> &q2, std::queue<Matrix> &q3,
2                       std::vector<ResultTime> &timeResultArr,
3                       std::vector<MatrixState> &matrixStateArr,
4                       bool &q1IsEmpty, bool &q2IsEmpty) {
5       std::chrono::time_point<std::chrono::system_clock> timeStart, timeEnd;
```

```
    int taskNum = 1;

    while (true) {
        if (!q2.empty()) {
            if (matrixStateArr[taskNum - 1].stage_1) {
                timeStart = std::chrono::system_clock::now();
                stage2(std::ref(q2), std::ref(q3));
                timeEnd = std::chrono::system_clock::now();

                saveResult(timeResultArr, timeStart, timeEnd,
                    timeResultArr[0].timeBegin, taskNum, 2);

                matrixStateArr[taskNum - 1].stage_2 = true;
                taskNum++;
            }
        } else if (q1IsEmpty) {
            break;
        }
    }

    q2IsEmpty = true;
}
```

Листинг 3.5 — Функция 3-ой ленты конвейерной обработки матрицы

```
void parallelStage3(std::queue<Matrix> &q3,
                    std::vector<ResultTime> &time_result_arr,
                    std::vector<MatrixState> &matrixStateArr,
                    bool &q2IsEmpty, int cntMatr, bool matrIsPrint) {
    std::chrono::time_point<std::chrono::system_clock> timeStart, timeEnd;
    int taskNum = 1;

    while (true) {
        if (!q3.empty()) {
            if (matrixStateArr[taskNum - 1].stage_2) {
                timeStart = std::chrono::system_clock::now();
                stage3(std::ref(q3), taskNum, cntMatr, matrIsPrint);
                timeEnd = std::chrono::system_clock::now();

                saveResult(time_result_arr, timeStart, timeEnd,
                    time_result_arr[0].timeBegin, taskNum, 3);

                matrixStateArr[taskNum - 1].stage_3 = true;
                taskNum++;
            }
        } else if (q2IsEmpty) {
            break;
```

```
22            }
23        }
24    }
```

Листинг 3.6 — Функция реализации 1-ого этапа обработки матрицы

```
1  void stage1(std::queue<Matrix> &q1, std::queue<Matrix> &q2) {
2      std::mutex m;
3
4      m.lock();
5      Matrix matrix = q1.front();
6      m.unlock();
7
8      matrix.minElem = matrix.getMinElem();
9
10     m.lock();
11     q2.push(matrix);
12     m.unlock();
13
14     m.lock();
15     q1.pop();
16     m.unlock();
17 }
```

Листинг 3.7 — Функция реализации 2-ого этапа обработки матрицы

```
1  void stage2(std::queue<Matrix> &q2, std::queue<Matrix> &q3) {
2      std::mutex m;
3
4      m.lock();
5      Matrix matrix = q2.front();
6      m.unlock();
7
8      matrix.modByMinElem();
9
10     m.lock();
11     q3.push(matrix);
12     m.unlock();
13
14     m.lock();
15     q2.pop();
16     m.unlock();
17 }
```

Листинг 3.8 — Функция реализации 3-ого этапа обработки матрицы

```
1  void stage3(std::queue<Matrix> &q3, int taskNum, int cntMatr, bool
       matrIsPrint) {
```

```
2       std::mutex m;
3
4       m.lock();
5       Matrix matrix = q3.front();
6       m.unlock();
7
8       matrix.sumElem = matrix.getSumElements();
9
10      m.lock();
11      q3.pop();
12      m.unlock();
13  }
```

3.3 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

3.4 Вывод

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

4 Экспериментальный раздел

4.1 Технические характеристики

Тестирование выполнялось на устройстве со следующими техническими характеристиками:

— Операционная система Ubuntu 21.04;

— Память 16 GiB (4,5 GiB выделено для нужд графического ядра)

— Процессор AMD® Ryzen 5 5500u with radeon graphics × 12

4.2 Время выполнения алгоритмов

Для замеров времени использовалась стандартная функция языка *std::chrono::system_clock::now()* [5]. Данная функция возвращает точку времени, представляющую текущее время системы. суммарное процессорное время, использованное программой. На листинге 4.1 показан способ применения данной функции при замерах.

Листинг 4.1 — Замер времени функции

```
timeStart = std::chrono::system_clock::now();
stage1(std::ref(q1), std::ref(q2));
timeEnd = std::chrono::system_clock::now();
```

Из результатов работы программы для 5 матриц размером 1000, представленных на листинге 4.2, можно сделать вывод, что конвейерная обработка работает согласно требованиям, выставленных в аналитической части.

Листинг 4.2 — Результаты работы программы для 5 матриц размером 1000

```
                    Линейная обработка | Конвейерная обработка
-------------------------------------------------------------------------
Матрица | Этап |   Начало   |    Конец   |   Начало   |    Конец
-------------------------------------------------------------------------
    1   |   1  | 0.000001   | 0.002897   | 0.000180   | 0.017111
    1   |   2  | 0.002897   | 0.009138   | 0.017116   | 0.029246
    1   |   3  | 0.009139   | 0.010996   | 0.029249   | 0.039317
-------------------------------------------------------------------------
    2   |   1  | 0.010996   | 0.014611   | 0.017113   | 0.030670
    2   |   2  | 0.014612   | 0.021266   | 0.030671   | 0.040434
    2   |   3  | 0.021266   | 0.023082   | 0.040437   | 0.049941
-------------------------------------------------------------------------
    3   |   1  | 0.023082   | 0.025945   | 0.030672   | 0.043867
    3   |   2  | 0.025945   | 0.032354   | 0.043868   | 0.053943
    3   |   3  | 0.032354   | 0.034182   | 0.053946   | 0.062653
-------------------------------------------------------------------------
```

4	1	0.034182	0.037137	0.043869	0.057170
4	2	0.037137	0.044486	0.057172	0.066339
4	3	0.044487	0.046599	0.066342	0.075127
5	1	0.046600	0.049689	0.057172	0.070995
5	2	0.049689	0.055959	0.070997	0.080095
5	3	0.055959	0.057649	0.080104	0.088796

На рисунке 4.1 представлена зависимость времени работы обработки 100 матриц от их размеров.

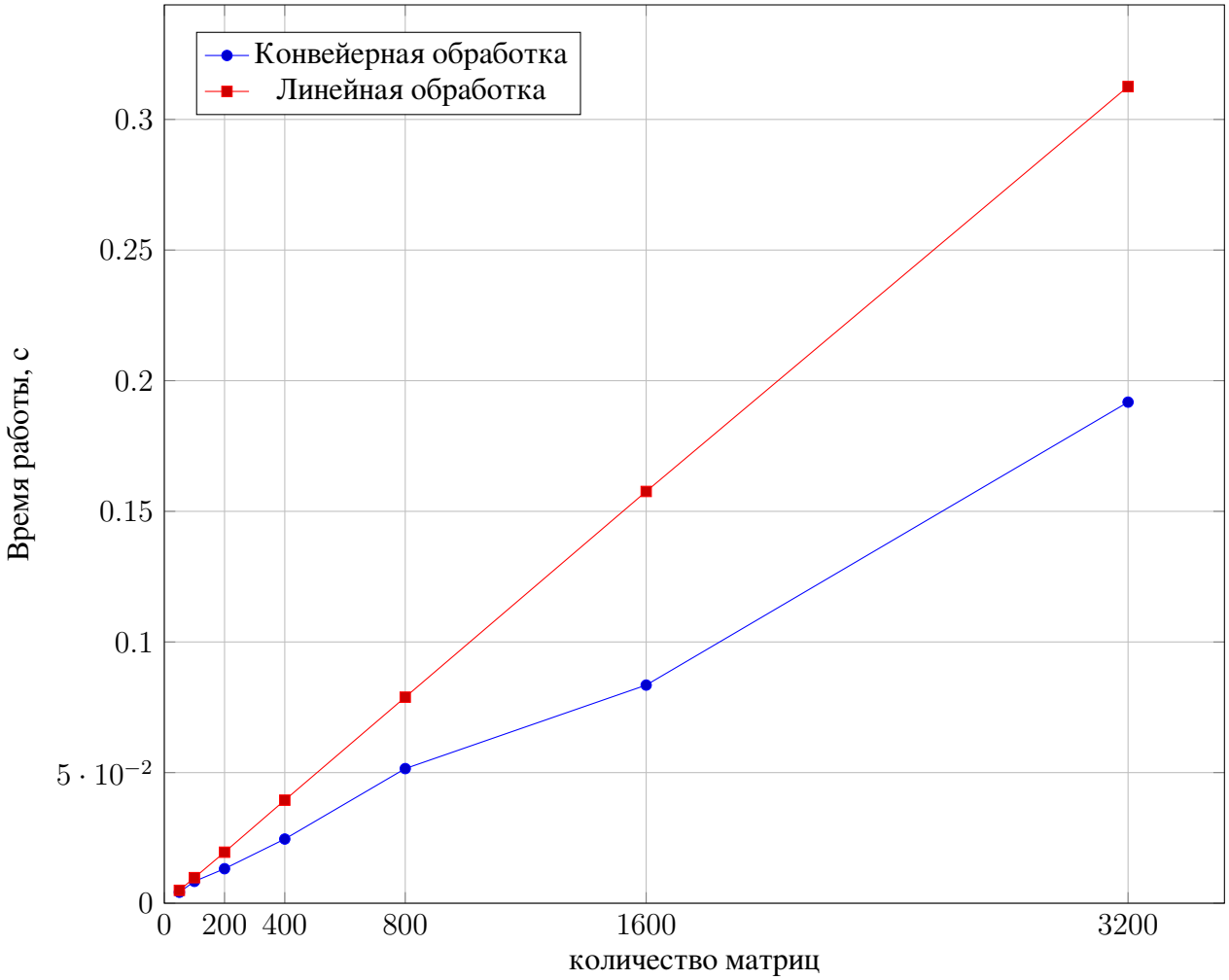

Рисунок 4.1 — Зависимость времени работы обработки матриц размера 100 от их количества

На рисунке 4.2 представлена зависимость времени работы обработки матриц размера 100 от их количества.

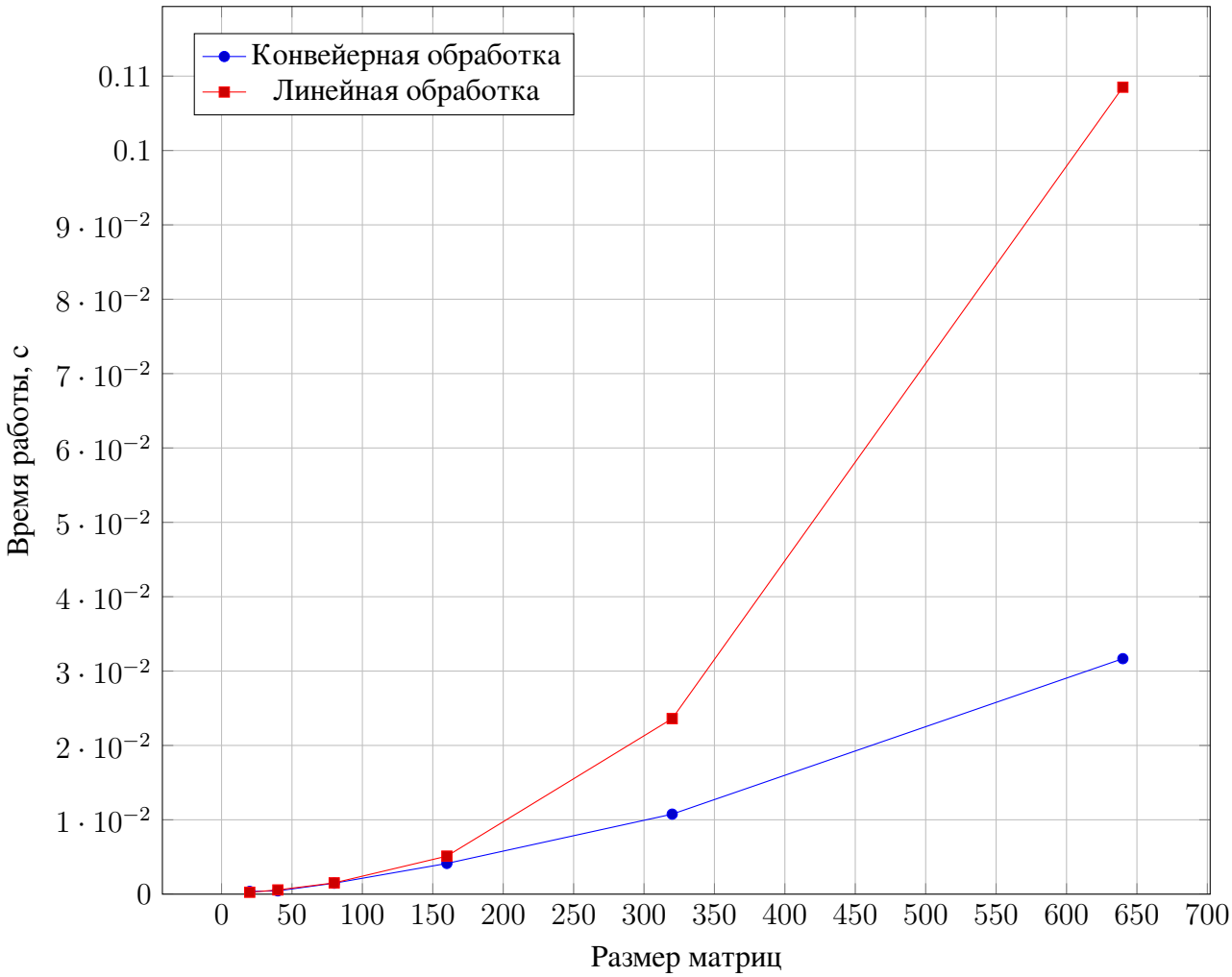

Рисунок 4.2 — Зависимость времени работы обработки 100 матриц от их размеров

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 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

Было экспериментально подтверждено различие во временной эффективности конвейерной и линейной реализаций обработак матриц. В результате исследований можно сделать вывод о том, что 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).

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 ЛИТЕРАТУРЫ

1. Конвейерная организация [Электронный ресурс]. — Режим доступа: http://www.citforum.mstu.edu.ru/hardware/svk/glava_5.shtml (дата обращения: 10.12.2021).

2. Страуструп Б. Программирование. Принципы и практика использования C++. — ООО ИД Вильямс, 2011. — ISBN: 9785457707351. — Режим доступа: https://books.google.ru/books?id=kttwBgAAQBAJ.

3. CMake [Электронный ресурс]. — Режим доступа: https://cmake.org/ (дата обращения: 10.10.2021).

4. CLion: A Cross-Platform IDE for C and C++ by JetBrains [Электронный ресурс]. — Режим доступа: https://www.jetbrains.com/clion/ (дата обращения: 10.10.2021).

5. std chrono system clock now [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/chrono/system_clock/now (дата обращения: 11.12.2021).