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 Динь Вьет Ань

Группа ИУ7И-54Б

Оценка (баллы)

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

Оглавление

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.

— Найдите в матрице первое число, большее среднего геометрического, и замените его наибольшим элементом матрицы. Если среднее антропометрическое значение не найдено или нет элемента, превышающего

среднее геометрическое, заменить элемент в первом столбце первой строки на наибольший элемент матрицы.

— Транспонировать матрицы.

— Найти произведение двух матриц.

Вывод

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

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.

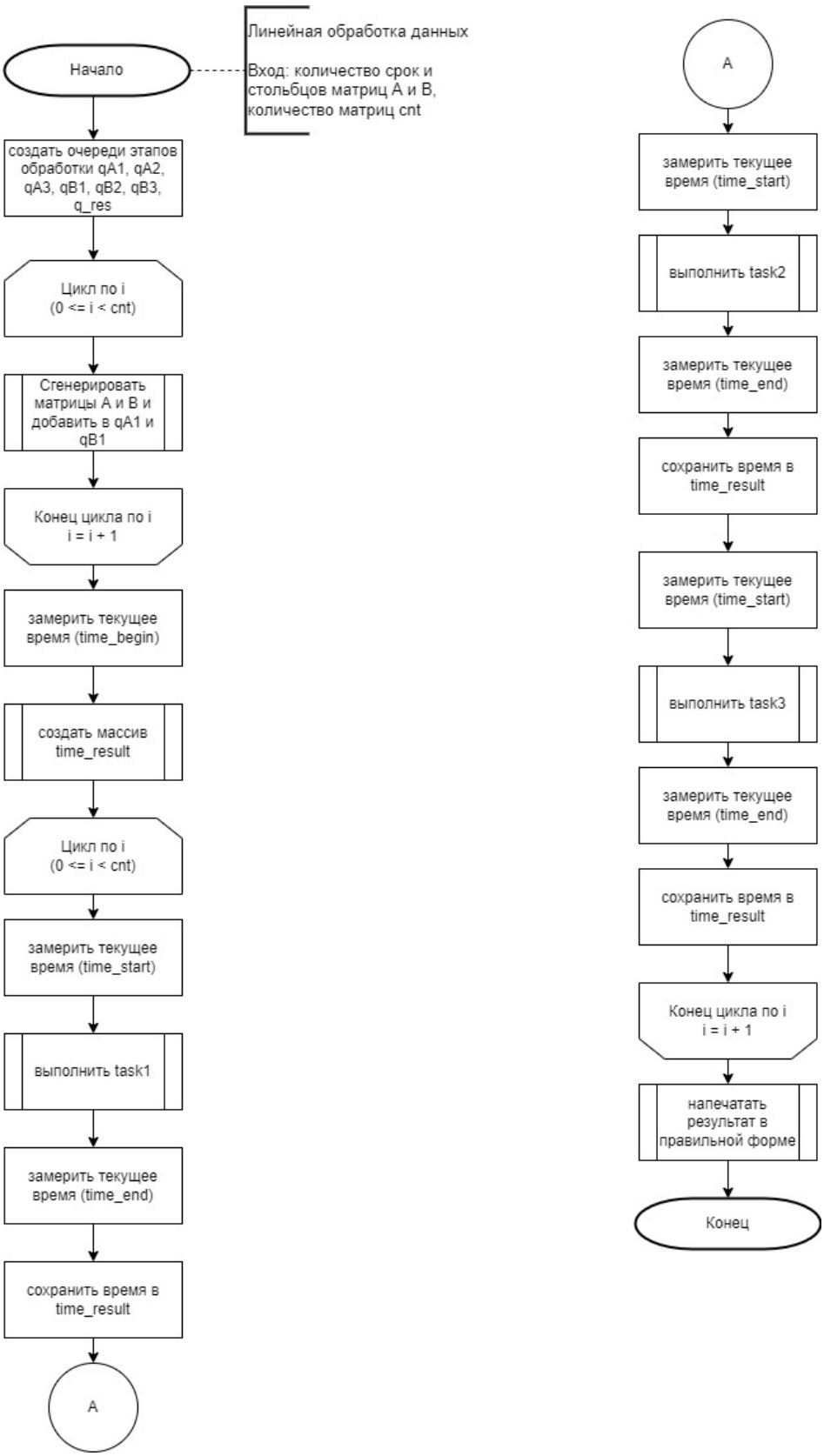

Рисунок 2.1 – Схема алгоритма линейной обработки матрицы

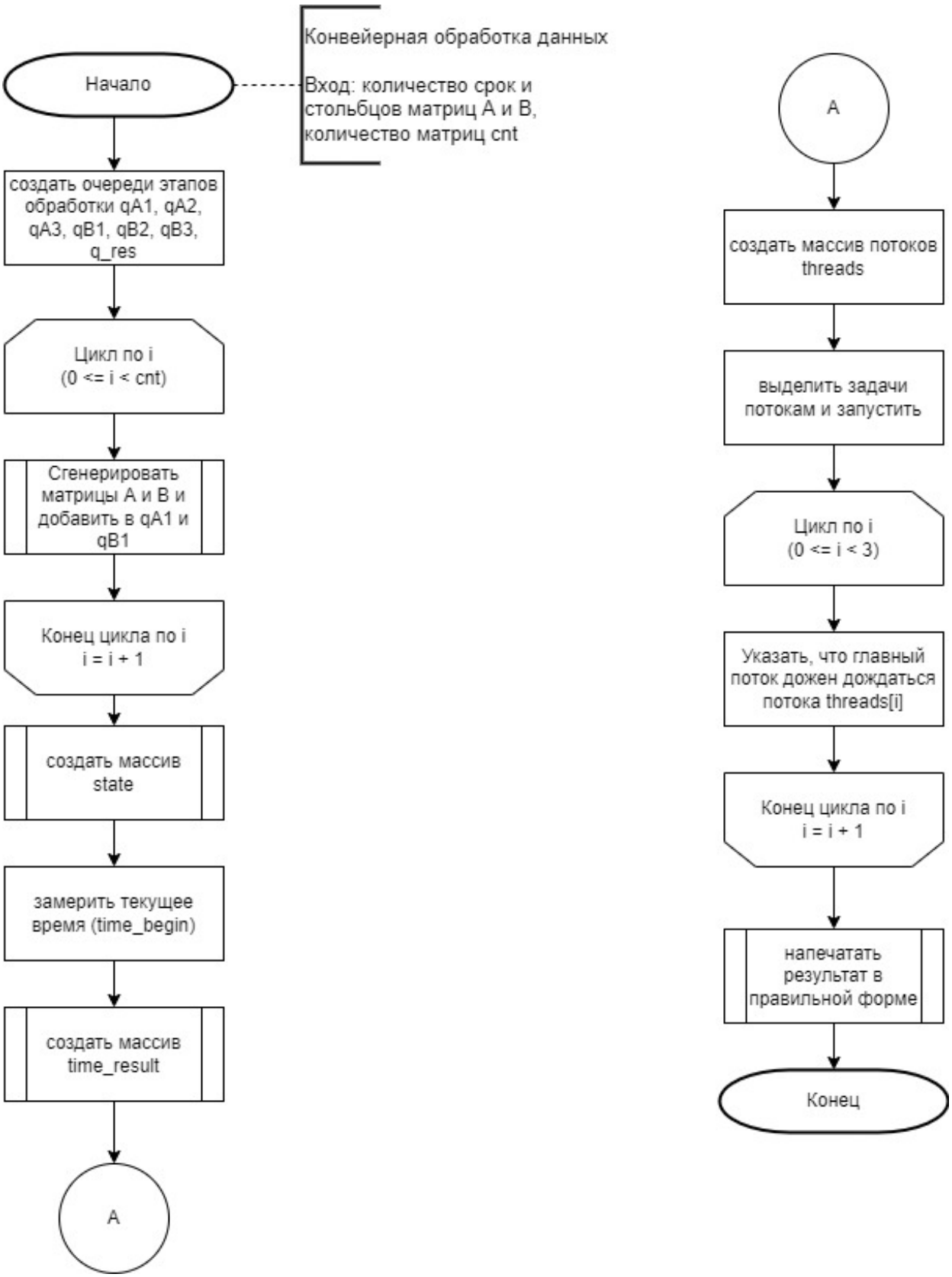

Рисунок 2.2 – Схема алгоритма конвейерной обработки матрицы

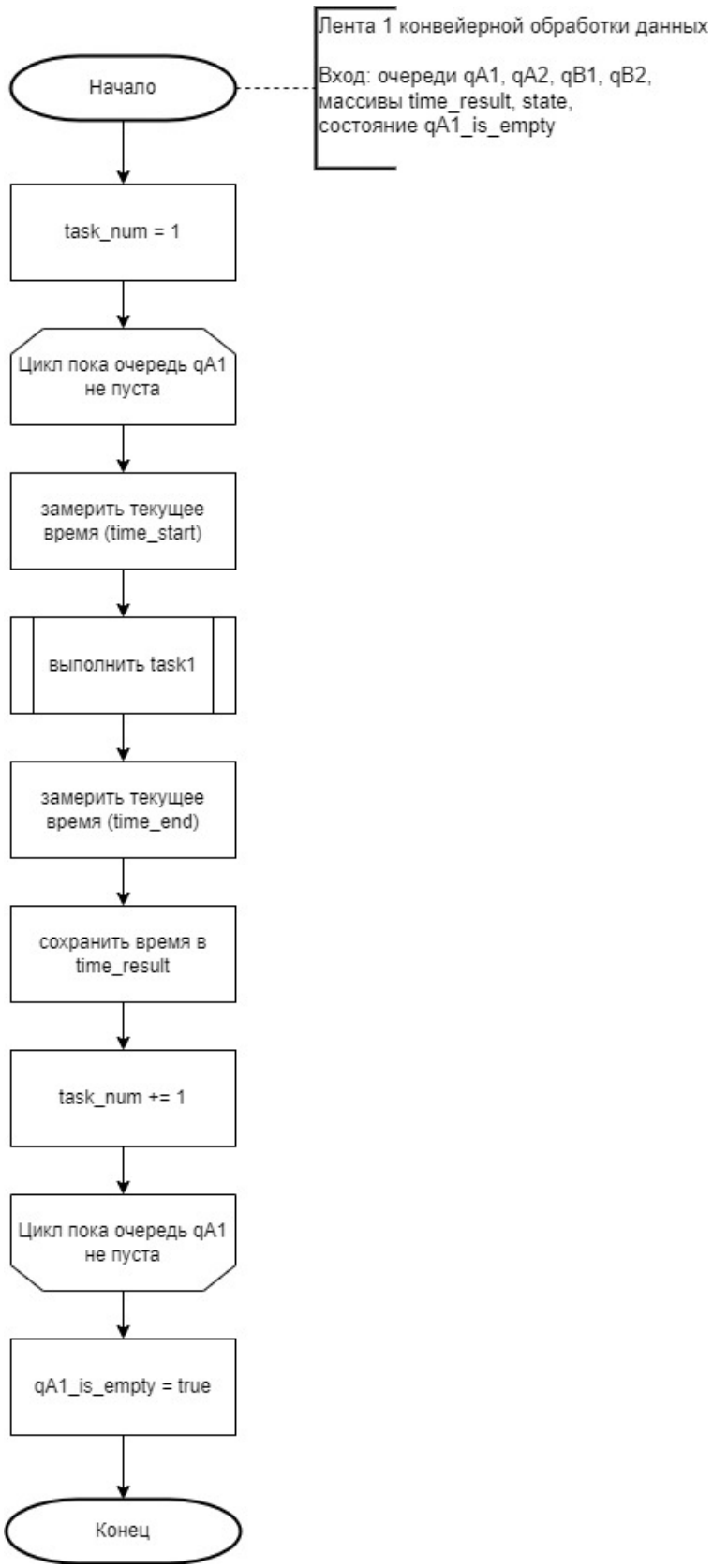

Рисунок 2.3 – Схема 1-ой ленты конвейерной обработки матрицы

Рисунок 2.4 – Схема 2-ой ленты конвейерной обработки матрицы

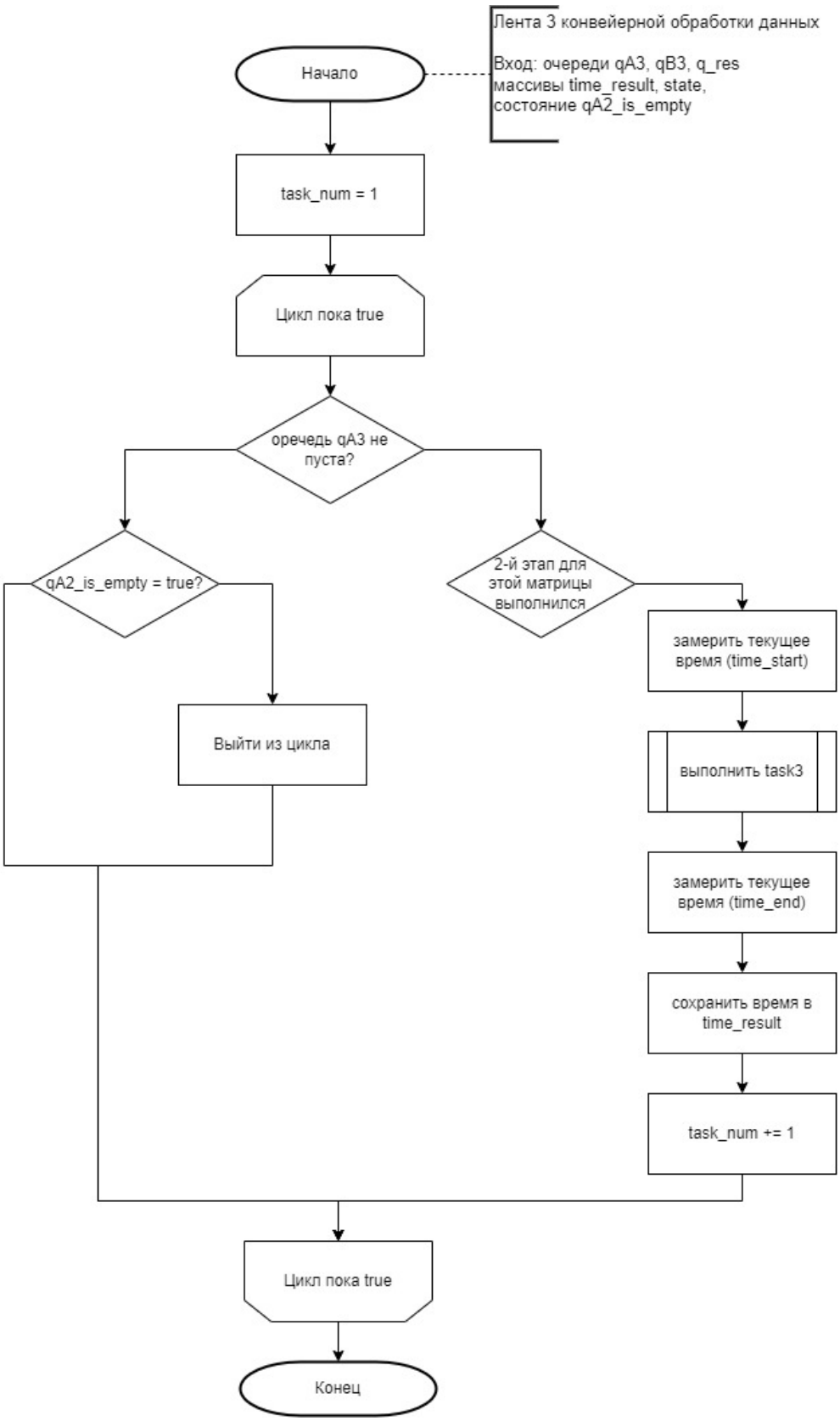

Рисунок 2.5 – Схема 3-ей ленты конвейерной обработки матрицы

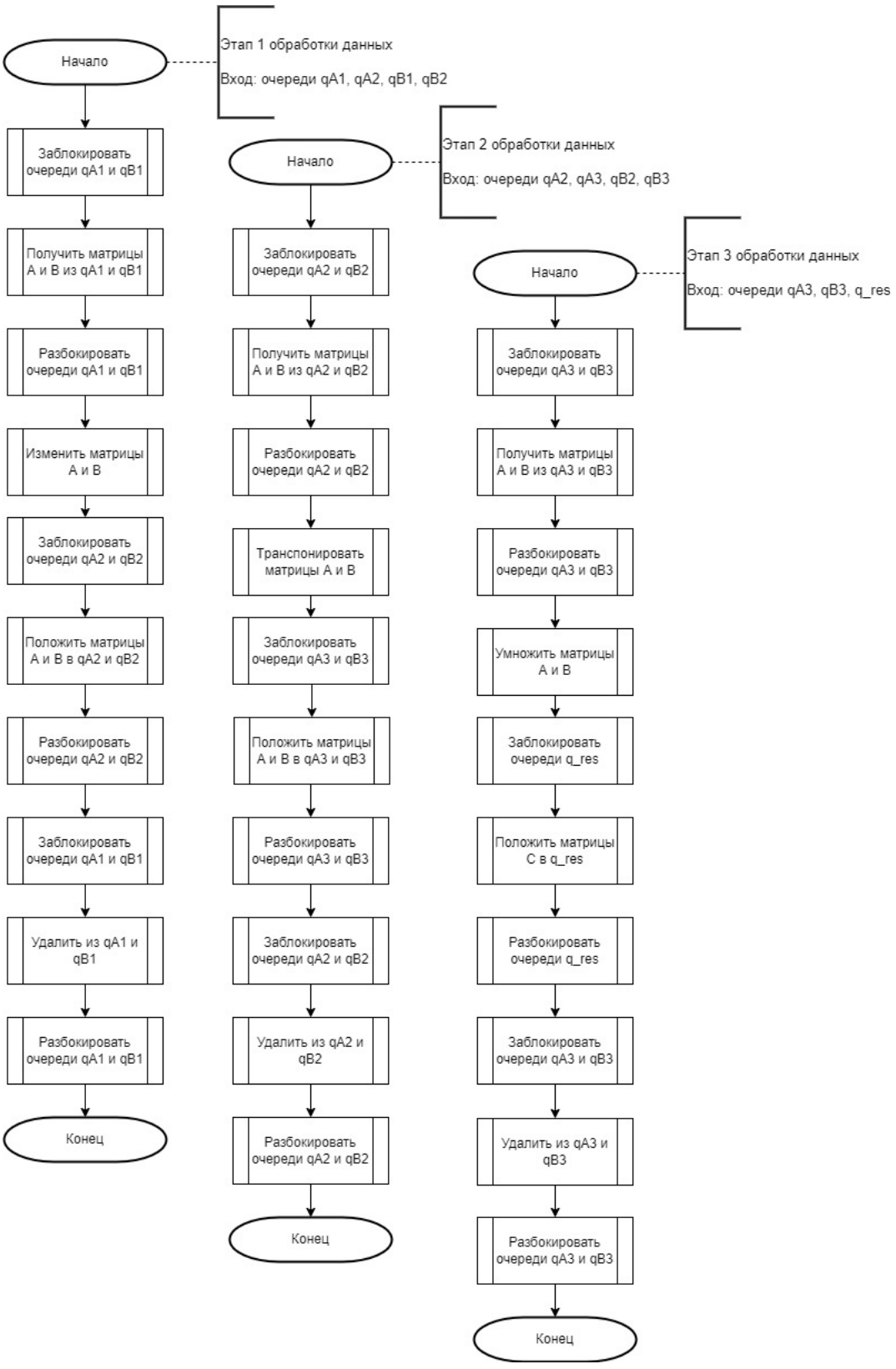

Рисунок 2.6 – Схема реализаций этапов обработки мматрицы

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;

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(int size, int cnt, bool is_count)
2  {
3      queue<matrix_t> qA1;
4      queue<matrix_t> qB1;
5      queue<matrix_t> qA2;
6      queue<matrix_t> qB2;
7      queue<matrix_t> qA3;
8      queue<matrix_t> qB3;
9      queue<matrix_t> q_res;
10
11     for (int i = 0; i < cnt; i++)
12     {
13         matrix_t matrA = generate_matrix(size, size);
14         matrix_t matrB = generate_matrix(size, size);
15         qA1.push(matrA);
16         qB1.push(matrB);
17     }
18     bool qA1_is_empty = false;
19     bool qA2_is_empty = false;
20     vector<matrix_state_t> state(cnt);
21     for (int i = 0; i < cnt; i++)
22     {
23         matrix_state_t tmp_state;
24         tmp_state.stage_1 = false;
25         tmp_state.stage_2 = false;
26         tmp_state.stage_3 = false;
27         state[i] = tmp_state;
28     }
29     chrono::time_point<std::chrono::system_clock> time_begin =
           chrono::system_clock::now();
30     vector<res_time_t> time_result_arr;
31     init_time_result_arr(time_result_arr, time_begin, cnt, 3);
32     thread threads[3];
33     threads[0] = thread(parallel_stage_1, ref(qA1), ref(qA2),
           ref(qB1), ref(qB2), ref(state), ref(time_result_arr),
           ref(qA1_is_empty));
34     threads[1] = thread(parallel_stage_2, ref(qA2), ref(qA3),
           ref(qB2), ref(qB3), ref(state), ref(time_result_arr),
           ref(qA1_is_empty), ref(qA2_is_empty));
35     threads[2] = thread(parallel_stage_3, ref(qA3), ref(qB3),
```

```
        ref(q_res), ref(state), ref(time_result_arr),
        ref(qA2_is_empty), cnt);
36      for (int i = 0; i < 3; i++)
37          threads[i].join();
38      if(is_count)
39          printf("      %4d       |      %4d       |    %.6f  \n",
40              size, cnt, time_result_arr[cnt - 1].end);
41      else
42          print_res_time(time_result_arr, cnt * 3);
43  }
```

Листинг 3.2 – Функция алгоритма линейной обработки матрицы

```
1   void liner(int size, int cnt, bool is_count)
2   {
3       queue<matrix_t> qA1;
4       queue<matrix_t> qB1;
5       queue<matrix_t> qA2;
6       queue<matrix_t> qB2;
7       queue<matrix_t> qA3;
8       queue<matrix_t> qB3;
9       queue<matrix_t> q_res;
10
11      std::chrono::time_point<std::chrono::system_clock>
            time_start, time_end,
12      time_begin = std::chrono::system_clock::now();
13
14      std::vector<res_time_t> time_result_arr;
15      init_time_result_arr(time_result_arr, time_begin, cnt, 3);
16
17      for (int i = 0 ; i < cnt; i++)
18      {
19          matrix_t matrA = generate_matrix(size, size);
20          matrix_t matrB = generate_matrix(size, size);
21          qA1.push(matrA);
22          qB1.push(matrB);
23      }
24      for (int i = 0; i < cnt; i++)
25      {
26          time_start = chrono::system_clock::now();
27          task1(ref(qA1), ref(qA2), ref(qB1), ref(qB2));
28          time_end = std::chrono::system_clock::now();
```

```
29          save_result(time_result_arr, time_start, time_end,
               time_result_arr[0].time_begin, i + 1, 1);
30          time_start = chrono::system_clock::now();
31          task2(ref(qA2), ref(qA3), ref(qB2), ref(qB3));
32          time_end = std::chrono::system_clock::now();
33          save_result(time_result_arr, time_start, time_end,
               time_result_arr[0].time_begin, i + 1, 2);
34          time_start = chrono::system_clock::now();
35          task3(ref(qA3), ref(qB3), ref(q_res));
36          time_end = std::chrono::system_clock::now();
37          save_result(time_result_arr, time_start, time_end,
               time_result_arr[0].time_begin, i + 1, 3);
38      }
39      if(is_count)
40          printf("␣␣␣␣␣␣%4d␣␣␣␣␣␣␣|␣␣␣␣␣␣%4d␣␣␣␣␣␣␣|␣␣␣%.6f␣␣\n",
41              size, cnt, time_result_arr[cnt - 1].end);
42      else
43          print_res_time(time_result_arr, cnt * 3);
44  }
```

Листинг 3.3 – Функция 1-ой ленты конвейерной обработки матрицы

```
1  void parallel_stage_1(queue<matrix_t> &qA1, queue<matrix_t> &qA2,
       queue<matrix_t> &qB1, queue<matrix_t> &qB2,
       vector<matrix_state_t> &state, vector<res_time_t>
       &time_result_arr, bool &qA1_is_empty)
2  {
3      chrono::time_point<chrono::system_clock> time_start, time_end;
4      int task_num = 1;
5      while(!qA1.empty())
6      {
7          time_start = chrono::system_clock::now();
8          task1(qA1, qA2, qB1, qB2);
9          time_end = std::chrono::system_clock::now();
10         save_result(time_result_arr, time_start, time_end,
               time_result_arr[0].time_begin, task_num, 1);
11         state[task_num - 1].stage_1 = true;
12         task_num ++;
13     }
14     qA1_is_empty = true;
15 }
```

Листинг 3.4 – Функция 2-ой ленты конвейерной обработки матрицы

```
void parallel_stage_2(queue<matrix_t> &qA2, queue<matrix_t> &qA3,
    queue<matrix_t> &qB2, queue<matrix_t> &qB3,
    vector<matrix_state_t> &state, vector<res_time_t>
    &time_result_arr, bool &qA1_is_empty, bool &qA2_is_empty)
{
    chrono::time_point<chrono::system_clock> time_start, time_end;
    int task_num = 1;
    while(1)
    {
        if(qA2.empty() == false)
        {
            if(state[task_num - 1].stage_1 == true)
            {
                time_start = chrono::system_clock::now();
                task2(qA2, qA3, qB2, qB3);
                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);

                state[task_num - 1].stage_2 = true;
                task_num ++;
            }
        }
        else if(qA1_is_empty)
                break;
    }
    qA2_is_empty = true;
}
```

Листинг 3.5 – Функция 3-ей ленты конвейерной обработки матрицы

```
void parallel_stage_3(queue<matrix_t> &qA3, queue<matrix_t> &qB3,
    queue<matrix_t> &q_res, vector<matrix_state_t> &state,
    vector<res_time_t> &time_result_arr, bool &qA2_is_empty, int
    cnt)
{
    chrono::time_point<chrono::system_clock> time_start, time_end;
    int task_num = 1;
    while(1)
```

```
6       {
7           if(qA3.empty() == false)
8           {
9               if(state[task_num - 1].stage_2 == true)
10              {
11                  time_start = chrono::system_clock::now();
12                  task3(qA3, qB3, q_res);
13                  time_end = std::chrono::system_clock::now();
14                  save_result(time_result_arr, time_start,
                        time_end, time_result_arr[0].time_begin,
                        task_num, 3);
15                  state[task_num - 1].stage_3 = true;
16                  task_num ++;
17              }
18          }
19          else if(qA2_is_empty)
20              break;
21      }
22  }
```

Листинг 3.6 – Функция реализации 1-ого этапа обработки матрицы

```
1  void task1(queue<matrix_t> &qA1, queue<matrix_t> &qA2,
       queue<matrix_t> &qB1, queue<matrix_t> &qB2)
2  {
3      mutex m;
4      m.lock();
5      matrix_t tmp_matrixA = qA1.front();
6      matrix_t tmp_matrixB = qB1.front();
7      m.unlock();
8      tmp_matrixA = change_matrix(tmp_matrixA);
9      tmp_matrixB = change_matrix(tmp_matrixB);
10     m.lock();
11     qA2.push(tmp_matrixA);
12     qB2.push(tmp_matrixB);
13     m.unlock();
14     m.lock();
15     qA1.pop();
16     qB1.pop();
17     m.unlock();
18 }
```

Листинг 3.7 – Функция реализации 2-ого этапа обработки матрицы

```
void task2(queue<matrix_t> &qA2, queue<matrix_t> &qA3,
    queue<matrix_t> &qB2, queue<matrix_t> &qB3)
{
    mutex m;
    m.lock();
    matrix_t tmp_matrixA = qA2.front();
    matrix_t tmp_matrixB = qB2.front();
    m.unlock();
    tmp_matrixA = truncate_matrix(tmp_matrixA);
    tmp_matrixB = truncate_matrix(tmp_matrixB);
    m.lock();
    qA3.push(tmp_matrixA);
    qB3.push(tmp_matrixB);
    m.unlock();
    m.lock();
    qA2.pop();
    qB2.pop();
    m.unlock();
}
```

Листинг 3.8 – Функция реализации 3-его этапа обработки матрицы

```
void task3(queue<matrix_t> &qA3, queue<matrix_t> &qB3,
    queue<matrix_t> &q_res)
{
    mutex m;
    m.lock();
    matrix_t tmp_matrixA = qA3.front();
    matrix_t tmp_matrixB = qB3.front();
    m.unlock();
    matrix_t tmp_matrix = mul_matrix(tmp_matrixA, tmp_matrixB);
    m.lock();
    q_res.push(tmp_matrix);
    m.unlock();
    m.lock();
    qA3.pop();
    qB3.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

Вывод

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

4 Исследовательская часть

В данном разделе приводятся результаты замеров затрат реализаций алгоритмов по процессорному времени.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

— операционная система Window 10 Home Single Language;

— память 8 Гб;

— процессор 11th Gen Intel(R) Core(TM) i5-1135G7 2.42 ГГц, 4 ядра.

Во время замера устройство было подключено к сети электропитания, нагружено приложениями окружения и самой системой замера.

4.2 Время выполнения реализаций алгоритмов

Замеры времени для каждой длины входного массива полигонов проводились 100 раз. В качестве результата взято среднее время работы алгоритма на данной длине.

В таблице 4.1 приведены результаты замеров времени работы реализаций линейного и конвейерного алгоритмов с одном размером матрицы.

Таблица 4.1 – Результат замеров времени (с)

Размер	Количество матрицы	Линейный	Конвейерный
100x100	50	0.238	0.008
100x100	100	0.426	0.011
100x100	200	0.833	0.029
100x100	400	1.638	0.077
100x100	800	3.354	0.138
100x100	1600	6.909	0.276

В таблице 4.2 приведены результаты замеров времени работы реализаций линейного и конвейерного алгоритмов с одном количеством матрицы.

Таблица 4.2 – Результат замеров времени (с)

Размер	Количество матрицы	Линейный	Конвейерный
20x20	100	0.266	0.002
40x40	100	0.062	0.005
80x80	100	0.305	0.011
160x160	100	1.900	0.032
320x320	100	11.791	0.127

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 21 раза при 400 матрицах, в 25 раза при 800 и в 25 при 1600). Так же конвейерная обработка показала себя лучше при увеличении размеров обрабатываемых матриц (в 60 раза при размере матриц 160x160 и в 86 раза при размере 320x320). 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 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 25 раза, а при матрицах 320x320 быстрее в 86 раза).

В ходе выполнения данной лабораторной работы были решены все 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.

Список использованных источников

1. Конвейерная организация [Электронный ресурс]. — URL: http://www.citforum.mstu.edu.ru/hardware/svk/glava_5.shtml (дата обращения: 10.04.2023)

2. C++ — Типизированный язык программирования / Хабр [Электронный ресурс]. — URL: https://habr.com/ru/hub/cpp/ (дата обращения: 10.04.2023).

3. chrono::system_clock::now — cppreference.com [Электронный ресурс]. — URL: https://en.cppreference.com/w/cpp/chrono/system_clock/now (дата обращения: 10.04.2023).

4. Intel [Электронный ресурс]. — URL: https://www.intel.ru/content/www/ru/ru/products/details/processors/core/i5.html (дата обращения: 10.04.2023).