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 орбаботка данных

Студент Цветков И.А.

Группа ИУ7-53Б

Оценка (баллы)

Преподаватель Волкова Л. Л.

Москва — 2021 г.

Содержание

Введение **3**

1 Аналитическая часть **4**

1.1 Конвейерной обработка данных 4

1.2 Описание алгоритмов 4

1.3 Вывод 5

2 Конструкторская часть **6**

2.1 Описание используемых типов данных 6

2.2 Структура разрабатываемого ПО 6

2.3 Схемы алгоритмов 7

2.4 Классы эквивалентности при тестировании 13

2.5 Вывод 13

3 Технологическая часть **14**

3.1 Средства реализации 14

3.2 Листинги кода 14

3.3 Сведения о модулях программы 19

3.4 Функциональные тесты 20

3.5 Вывод 20

4 Исследовательская часть **21**

4.1 Технические характеристики 21

4.2 Демонстрация работы программы 21

4.3 Время выполнения алгоритмов 24

4.4 Вывод 28

Заключение **29**

Список литературы **30**

Введение

Использование параллельной обработки открывает новые способы для ускорения работы программ. Конвейрная обработка является одним из примеров, где использование принципов параллельности помогает ускорить обработку данных. Суть та же, что и при работе реальных конвейрных лент - материал (данное) поступает на обработку, после окончания обработки материал передается на место следующего обработчина, при этом предыдущий обработчик не ждет полного цикла обработки материала, а получает новый материал и работает с ним.

Целью данной работы является изучение принципов конвейрной обработки данных. Для достижения поставленной цели необходимо выполнить следующие задачи:

- изучить основы конвейрной обработки данных;
- описать алгоритмы обработки матрицы, которые будут использоваться в текущей лабораторной работе;
- привести схемы конвейрной и линейной обработок;
- описать используемые типы и структуры данных;
- описать структуру разрабатываемого программного обеспечения;
- реализовать разработанный алгоритм;
- провести функциональное тестирование разработанного алгоритма;
- провести сравнительный анализ по времени для реализованного алгоритма;
- подготовить отчет по лабораторной работе.

1 Аналитическая часть

В этом разделе будет представлена информация по поводу сути конвейерной обработки данных.

1.1 Конвейерной обработка данных

Конвейер [1] – организация вычислений, при которой увеличивается количество выполняемых инструкций за единицу времени за счет использования принципов параллельности.

Конвейеризация (или конвейерная обработка) в общем случае основана на разделении подлежащей исполнению функции на более мелкие части, называемые ступенями, и выделении для каждой из них отдельного блока аппаратуры. Так, обработку любой машинной команды можно разделить на несколько этапов (несколько ступеней), организовав передачу данных от одного этапа к следующему. При этом конвейерную обработку можно использовать для совмещения этапов выполнения разных команд. Производительность при этом возрастает, благодаря тому что одновременно на различных ступенях конвейера выполняется несколько команд. Конвейерная обработка такого рода широко применяется во всех современных быстродействующих процессорах.

Конвейеризация увеличивает 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, неконвейерной схемой.

1.2 Описание алгоритмов

В качестве примера для конвейерной обработки будет обрабатываться матрица. Всего будет использовано три ленты, которые делают следующее.

1. Находится среднее арифмитическое значений матрицы.
2. Находится максимальный элемент матрицы.
3. Нечетный элемент матрицы заменяется на среднее арифметическое матрицы, а четные - на макимальный элемент.

1.3 Вывод

В данном разделе было рассмотрено понятие конвейерной обработки, а также выбраны этапы для обработки матрицы, которые будут обрабатывать ленты конвейера.

Программа будет получать на вход количество задач (количество матриц), размер матрицы (используются только квадратные матрицы), а также выбор алгоритма – линейный или конвейерный. При неверном вводе какого-то из значений будет выдаваться сообщение об ошибке.

Реализуемое ПО дает возможность получить лог программы для N задач при линейной и конвейрной обработке. Также имеется возможность провести тестирование по времени для разного количества задач (матриц) и разных размеров самих матриц.

2 Конструкторская часть

В этом разделе будут представлено описание используемых типов данных, а также схемы алгоритмов конвейрной и линейной обработки матриц.

2.1 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие типы данных:

- количество задач (матриц) - целое число типа *int*;
- размер матрицы - целое число типа *int*;
- структура *matrix_s* - содержит информацию о матрице – сама матрицы (имеет тип std::vector<std::vector<double»), ее размер (size_t), а также хранит информацию о найденном среднем арифметическом и максимальном элементе;
- структура *queues_t* - содержит информацию об очередях (описывает их, каждая очредь имеет тип данных std::queue<matrix_t>).

2.2 Структура разрабатываемого ПО

В данном ПО будет реализован метод структурного программирования. Для взаимодействия с пользователем будет разработано меню, которое будет предоставлять возможность выбрать нужный алгоритм - линейный или конвейерный, замерить время и вывести информацию об этапах.

Для работы будут разработаны следующие процедуры:

- процедура, реализующая генерацию квадратной матрицы по ее размеру, входные данные - размер матрицы, выходные - полученная матрица;
- процедура вывода полученной после обработки матрицы на экран (для отладки), входные данные - матрица, выходные - выведенная н экран матрица;

- процедуры обработки матрицы (нахождение среднего арифметического, макимального элемента, а также вставка их в матрицу), входные данные - матрица (и среднее арифметическое и максимальный элемент - для заполнения матрица), выходные - среднее арифметическое (задача 1), максимальный элемент (задача 2), измененная матрица (задача 3);

- процедуры, реализующие линейную и конвейерную обработки этапов матрицы, входные данные - кол-во задач (матриц), размер матриц, выходные - лог работы программы с выводом времени;

- процедуры замера времени поэтапной орбаботки матрицы с использованием конвейера и линейного подхода, входные данные - количество матриц, начальный размер матрицы, конечный размер матрицы (если время замеряется для разных размеров матриц), размер матриц, начальное количество матриц, конечное количество матрицы (если время замеряется для разного количества задач (матриц)), выходные - результаты замеров времени;

- процедура для построения графиков по полученным временным замерам, входные данные - замеры времени, выходные - график.

2.3 Схемы алгоритмов

На рисунке 2.1 представлена схема алгоритма линейной обработки матрицы. На рисунках 2.2 схема алгоритма конвейрной обработки матрицы, а на рисунках 2.3-2.5 - схемы потоков обработки матрицы (ленты конвейера).

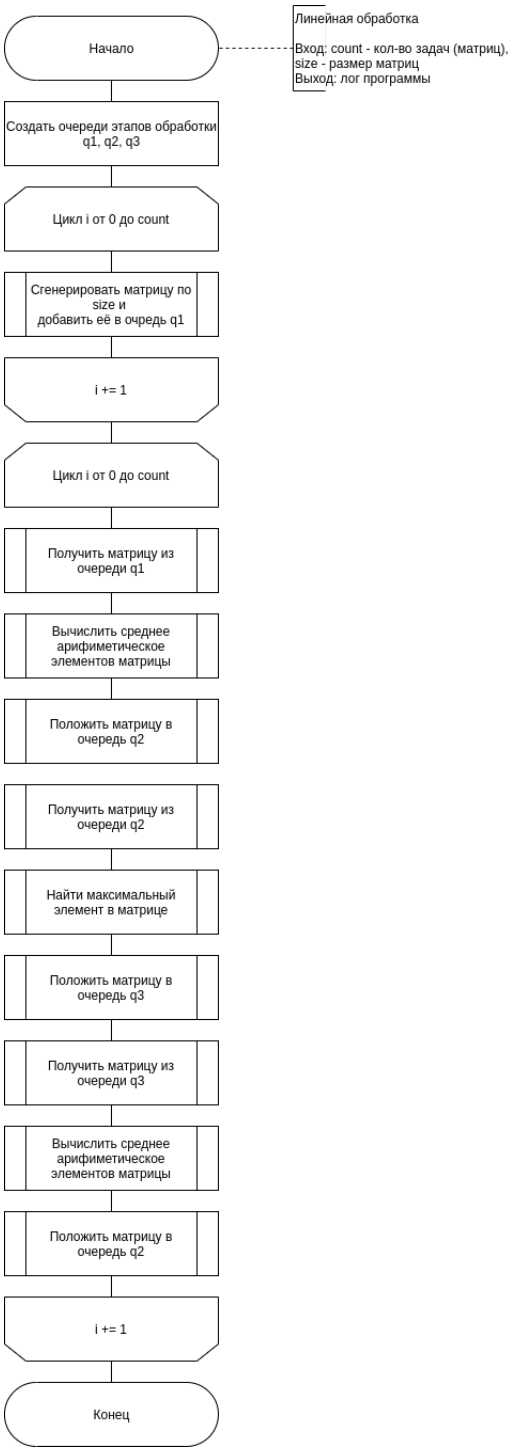

Рисунок 2.1 – Схема алгоритма линейной обработки матрицы

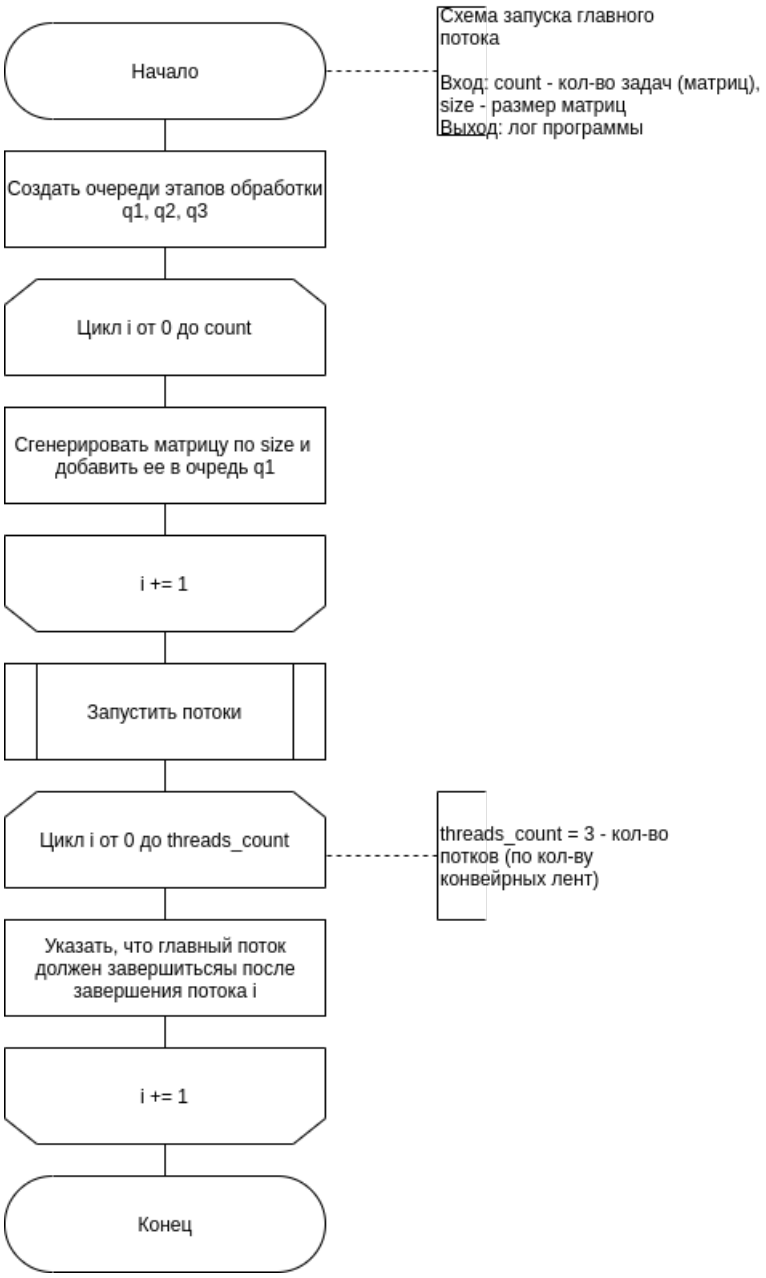

Рисунок 2.2 – Схема конвейрной обработки матрицы

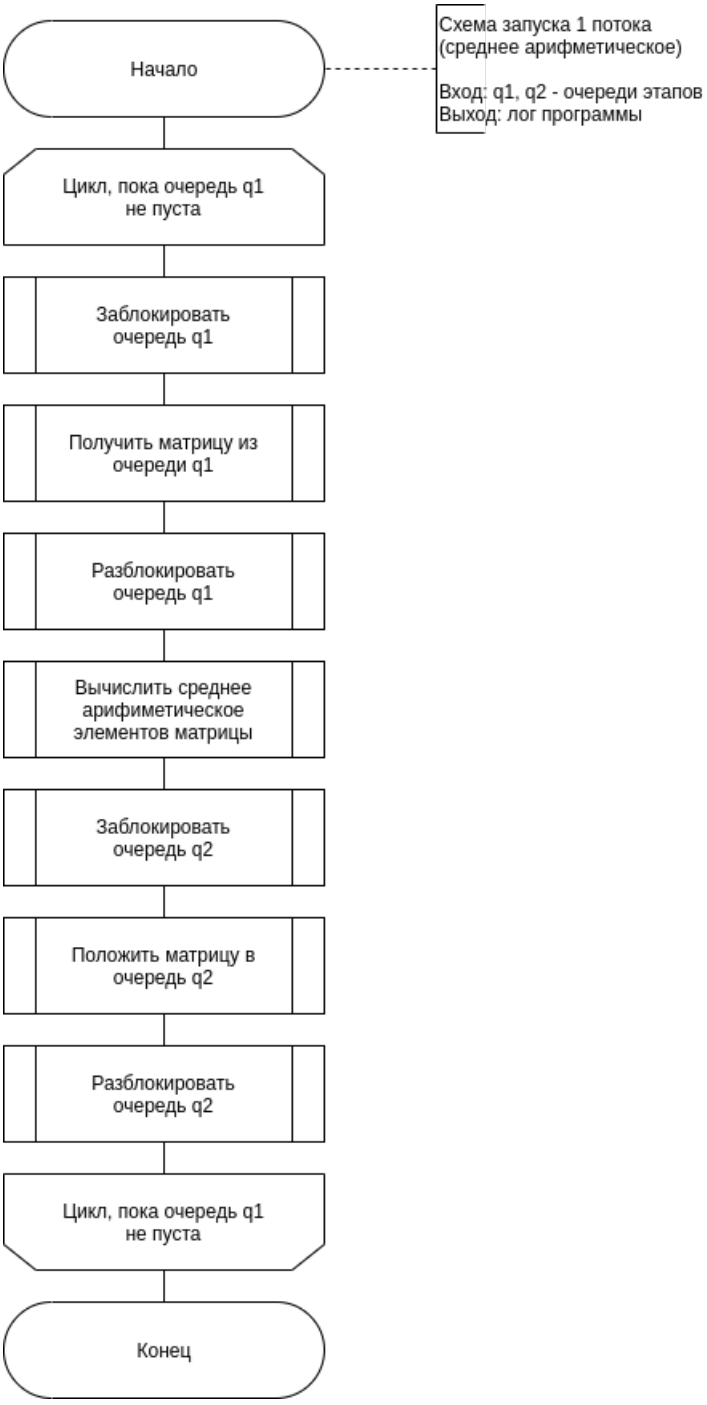

Рисунок 2.3 – Схема 1 потока обработки матрицы - нахождение среднего арифметического

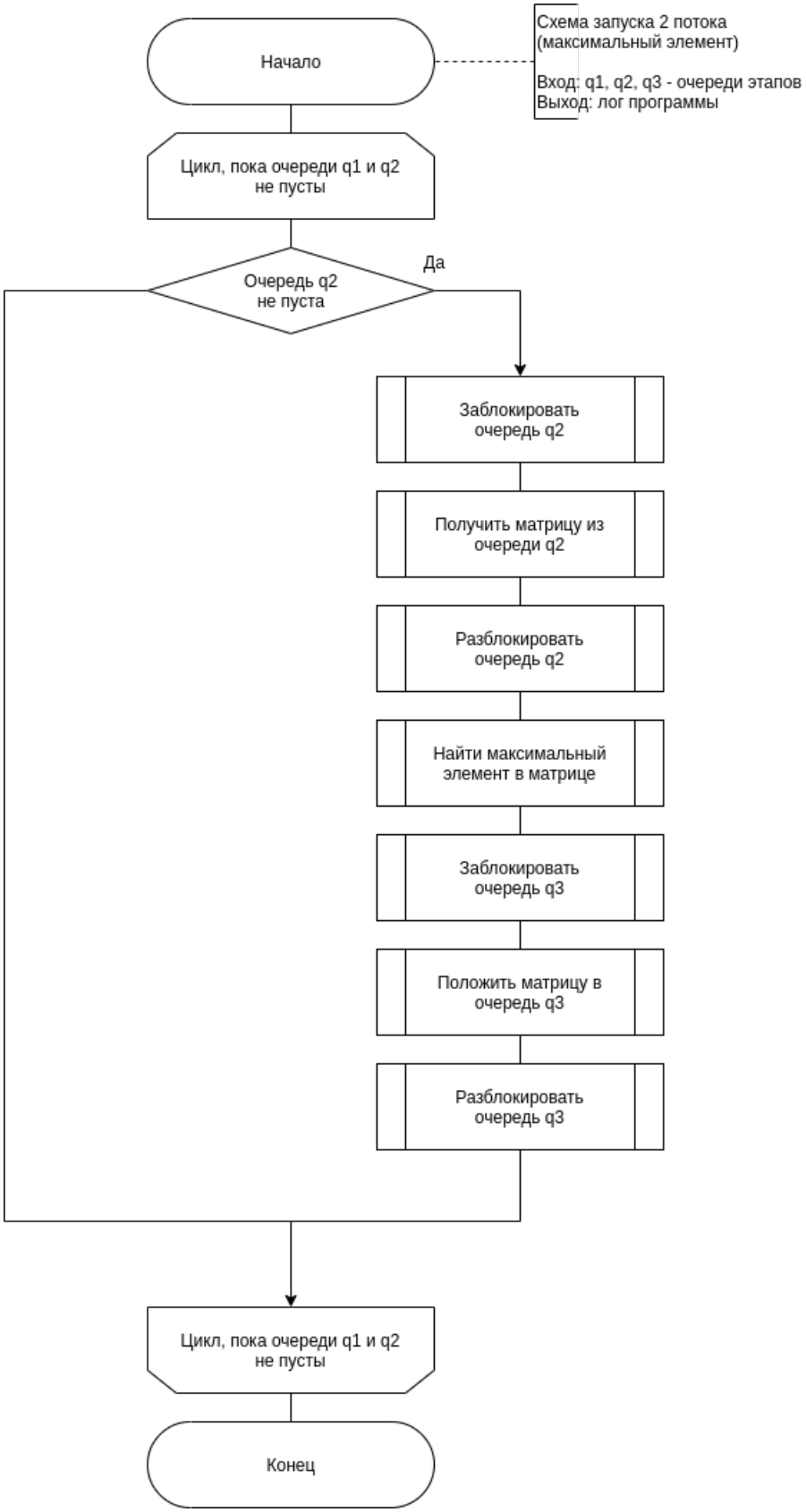

Рисунок 2.4 – Схема 2 потока обработки матрицы - нахождение максимального элемента

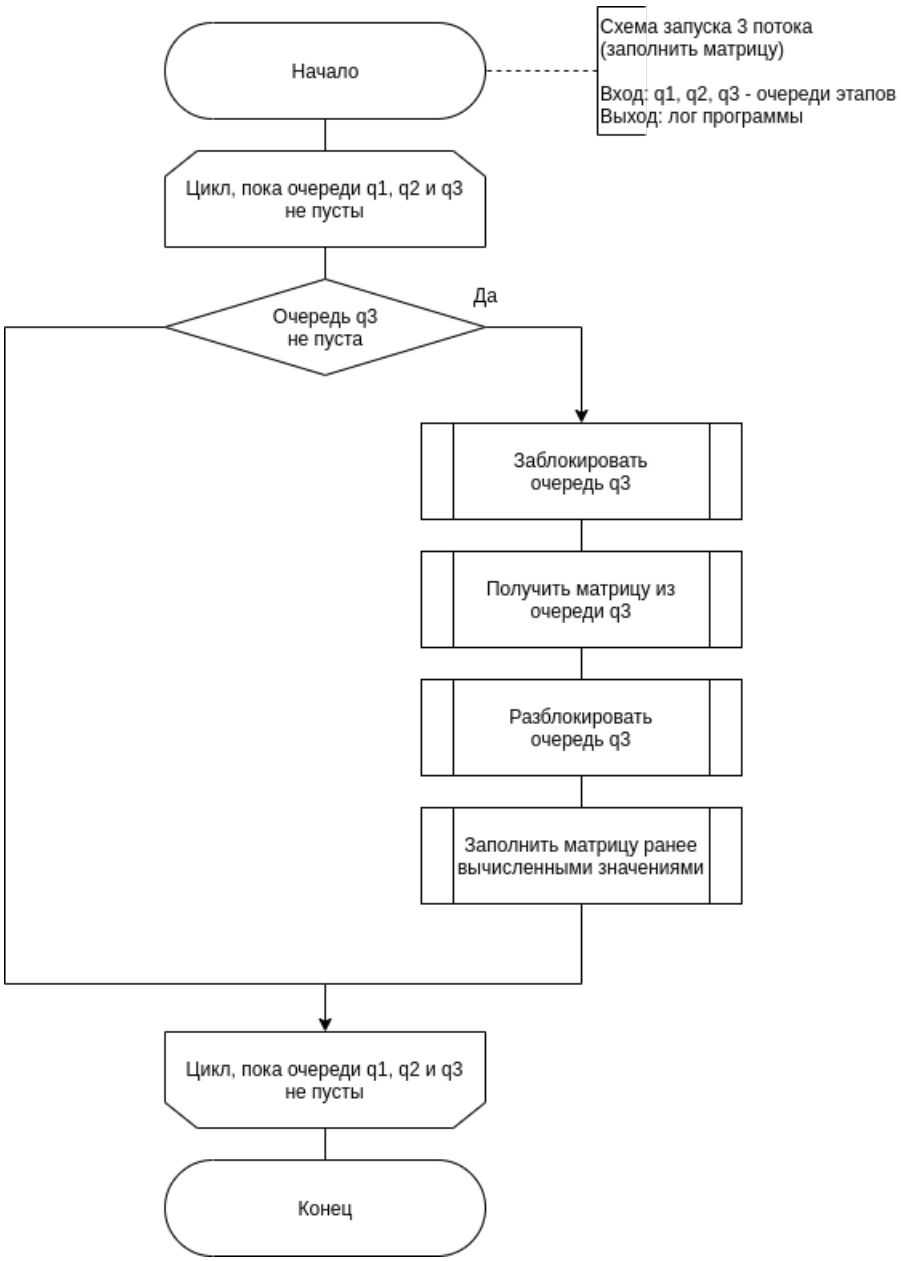

Рисунок 2.5 – Схема 3 потока обработки матрицы - заполнения матрицы новыми значенями

2.4 Классы эквивалентности при тестировании

Для тестирования выделены классы эквивалентности, представленные ниже.

1. Неверно выбран пункт меню - не число или число, меньшее 0 или большее 4.
2. Неверно введено количество матриц - не число или число, меньшее 1.
3. Неверно введен размер матриц - не число или число, меньшее 2.
4. Корректный ввод всех параметров.

2.5 Вывод

В данном разделе были построены схемы алгоритмов, рассматриваемых в лабораторной работе, были описаны классы эквивалентности для тестирования, структура программы.

3 Технологическая часть

В данном разделе будут рассмотрены средства реализации, а также представлены листинги линейной и конвейрной орбаботок матрицы.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования $C++$ [2]. В текущей лабораторной работе требуется замерить процессорное время для выполняемой программы, а также реализовать принципы многопоточного алгоритма. Все эти инструменты присутствуют в выбранном языке программирования.

Функции построения графиков были реализованы с использованием языка программирования *Python* [3].

Время замерено с помощью *std::chrono::system_clock::now(...)* - функции из библиотеки *chrono* [4].

3.2 Листинги кода

В листинге 3.1 представлены реализация алгоритма Брезенехема для построения отрезка, а в листингах 3.2-3.5 - алгоритмы для построения спектра отрезков по Брезенехему с многопоточностью и без нее соответственно.

Листинг 3.1 – Алгоритм линейной обработки матрицы

```
void parse_linear(int count, size_t size, bool is_print)
{
    time_now = 0;

    std::queue<matrix_t> q1;
    std::queue<matrix_t> q2;
    std::queue<matrix_t> q3;

    queues_t queues = {.q1 = q1, .q2 = q2, .q3 = q3};

    for (int i = 0; i < count; i++)
    {
        matrix_t res = generate_matrix(size);

        queues.q1.push(res);
    }

    for (int i = 0; i < count; i++)
    {
        matrix_t matrix = queues.q1.front();
        stage1_linear(matrix, i + 1, is_print);
        queues.q1.pop();
        queues.q2.push(matrix);

        matrix = queues.q2.front();
        stage2_linear(matrix, i + 1, is_print); // Stage 2
        queues.q2.pop();
        queues.q3.push(matrix);

        matrix = queues.q3.front();
        stage3_linear(matrix, i + 1, is_print); // Stage 3
        queues.q3.pop();
    }
}
```

Листинг 3.2 – Алгоритм конвейерной обработки матрицы

```
1  void parse_parallel(int count, size_t size, bool is_print)
2  {
3      time_now = 0;
4
5      std::queue<matrix_t> q1;
6      std::queue<matrix_t> q2;
7      std::queue<matrix_t> q3;
8
9      queues_t queues = {.q1 = q1, .q2 = q2, .q3 = q3};
10
11
12     for (int i = 0; i < count; i++)
13     {
14         matrix_t res = generate_matrix(size);
15
16         q1.push(res);
17     }
18
19     std::thread threads[THREADS];
20
21     threads[0] = std::thread(stage1_parallel, std::ref(q1),
           std::ref(q2), std::ref(q3), is_print);
22     threads[1] = std::thread(stage2_parallel, std::ref(q1),
           std::ref(q2), std::ref(q3), is_print);
23     threads[2] = std::thread(stage3_parallel, std::ref(q1),
           std::ref(q2), std::ref(q3), is_print);
24
25     for (int i = 0; i < THREADS; i++)
26     {
27         threads[i].join();
28     }
29 }
```

Листинг 3.3 – Алгоритм запуска 1 потока для нахождения среднего арифметического элементов матрицы

```
1  void stage1_parallel(std::queue<matrix_t> &q1, std::queue<matrix_t>
       &q2, std::queue<matrix_t> &q3, bool is_print)
2  {
3      int task_num = 1;
4
5      std::mutex m;
6
7      while(!q1.empty())
8      {
9          m.lock();
10         matrix_t matrix = q1.front();
11         m.unlock();
12
13         log(matrix, task_num++, 1, get_avg, is_print);
14
15         m.lock();
16         q2.push(matrix);
17         q1.pop();
18         m.unlock();
19     }
20 }
```

Листинг 3.4 – Алгоритм запуска 2 потока для нахождения максимального элемента матрицы

```
1  void stage2_parallel(std::queue<matrix_t> &q1, std::queue<matrix_t>
       &q2, std::queue<matrix_t> &q3, bool is_print)
2  {
3      int task_num = 1;
4
5      std::mutex m;
6
7      do
8      {
9          m.lock();
10         bool is_q2empty = q2.empty();
11         m.unlock();
12
13         if (!is_q2empty)
14         {
15             m.lock();
16             matrix_t matrix = q2.front();
17             m.unlock();
18
19             log(matrix, task_num++, 2, get_max, is_print);
20
21             m.lock();
22             q3.push(matrix);
23             q2.pop();
24             m.unlock();
25         }
26     } while (!q1.empty() || !q2.empty());
27 }
```

Листинг 3.5 – Алгоритм запуска 3 потока для заполнения матрицы вычисленными значениями

```
void stage3_parallel(std::queue<matrix_t> &q1, std::queue<matrix_t>
   &q2, std::queue<matrix_t> &q3, bool is_print)
{
    int task_num = 1;

    std::mutex m;

    do
    {
        m.lock();
        bool is_q3empty = q3.empty();
        m.unlock();

        if (!is_q3empty)
        {
            m.lock();
            matrix_t matrix = q3.front();
            q3.pop();
            m.unlock();

            log(matrix, task_num++, 3, fill_matrix, is_print);
        }

    } while (!q1.empty() || !q2.empty() || !q3.empty());
}
```

3.3 Сведения о модулях программы

Программа состоит из двух модулей:

- *main.cpp* - файл, содержащий меню программы;
- *matrix.h* и *matrix.cpp* - файлы, содержащие код работы с матрицей (генерация, печать и алгоритмы обработки матрицы);
- *conveyor.h* и *conveyor.cpp* - файлы, содержащие код реализации линейной и конвейерной обработок, а также функции замера времени;

- *graph_build.py* - файл, содержащий функции построения графиков для замеров по времени.

3.4 Функциональные тесты

В таблице 3.1 приведены тесты для функций программы. Тесты *для всех функций* пройдены успешно.

Таблица 3.1 – Функциональные тесты

Алгоритм	Кол-во матриц	Размер матриц	Ожидаемый результат
Конвейрная	-5	500	Сообщение об ошибке
Конвейрная	10	-23	Сообщение об ошибке
Линейная	50	1500	Вывод лога работы программы
Конвейрная	10	100	Вывод лога работы программы
Конвейрная	5	10	Вывод лога работы программы

3.5 Вывод

Были представлены листинги всех алгоритмов линейной и конвейерной обработки матриц. Также в данном разделе была приведена информации о выбранных средствах для разработки алгоритмов и сведения о модулях программы, проведено функциональное тестирование.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программа, а также проведен сравнительный анализ алгоритмов при различных ситуациях на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование представлены далее:

- операционная система: Ubuntu 20.04.3 [5] Linux [6] x86_64;
- память: 8 GiB;
- процессор: Intel® Core™ i5-7300HQ CPU @ 2.50GHz [7];
- 4 физических ядра, 4 логических ядра [7].

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

На рисунке 4.1 представлен результат работы программы при линейной обработке, а на рисунке 4.2 – при конвейерной.

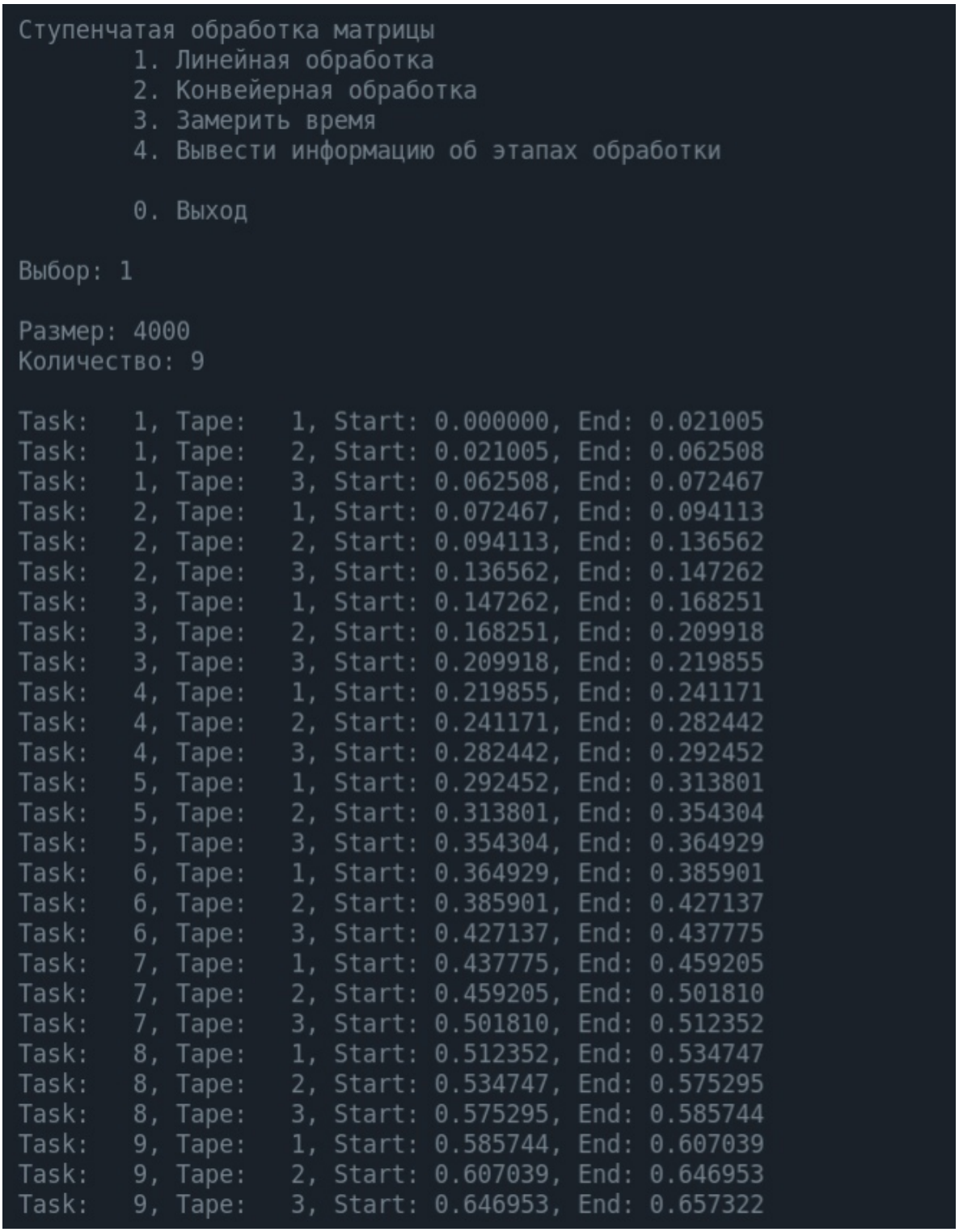

Рисунок 4.1 – Пример работы программы (линейная)

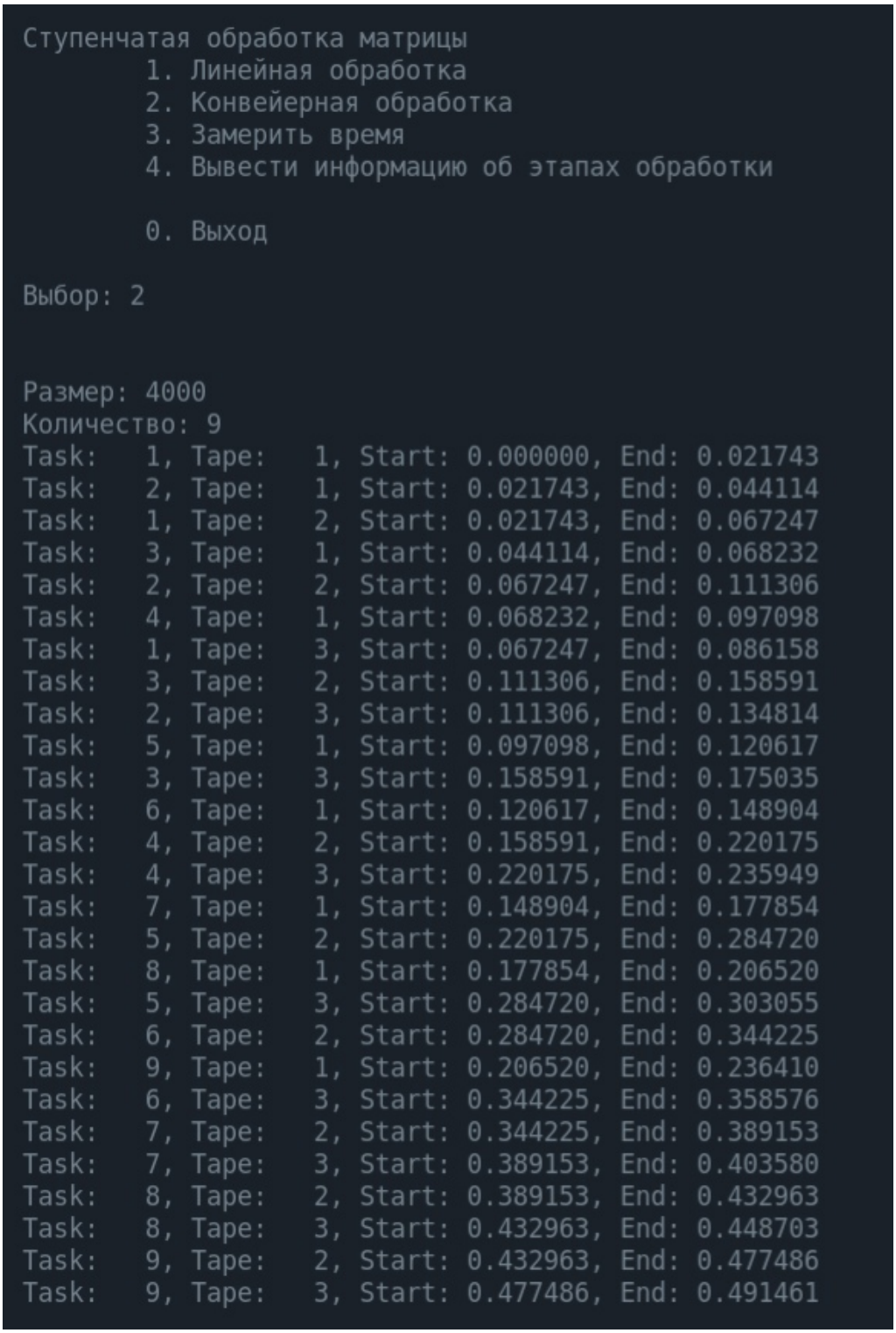

Рисунок 4.2 – Пример работы программы (конвейерная)

4.3 Время выполнения алгоритмов

Как было сказано выше, используется функция замера процессорного времени *std::chrono::system_clock::now(...)* из библиотеки *chrono* на C++. Функция возвращает процессорное время типа float в секундах.

Использовать функцию приходится дважды, затем из конечного времени нужно вычесть начальное, чтобы получить результат.

Замеры проводились для разного рамзера матриц, а также для разного количества матриц, чтобы определить, когда наиболее эффективно использовать конвейерную обработку.

Результаты замеров приведены в таблицах 4.1-4.4 (время в с).

Таблица 4.1 – Результаты замеров времени (линейная – разное кол-во матриц)

Кол-во матриц	Время
10	0.1847
15	0.2841
20	0.3771
25	0.4755
30	0.5648
35	0.6581
40	0.7558
45	0.8529
50	0.9818

Таблица 4.2 – Результаты замеров времени (конвейрная – разное кол-во матриц)

Кол-во матриц	Время
10	0.1432
15	0.1632
20	0.2603
25	0.3187
30	0.3943
35	0.3566
40	0.4431
45	0.5429
50	0.6472

Таблица 4.3 – Результаты замеров времени (линейная – разные размеры матриц)

Размер матриц	Время
1500	0.1574
1600	0.1797
1700	0.201
1800	0.2268
1900	0.2533
2000	0.2818
2100	0.3062
2200	0.337
2300	0.3736
2400	0.4045
2500	0.4369

Таблица 4.4 – Результаты замеров времени (конвейерная – разные размеры матриц)

Размер	Время
1500	0.0912
1600	0.0969
1700	0.1069
1800	0.1527
1900	0.1604
2000	0.1876
2100	0.1986
2200	0.2524
2300	0.2858
2400	0.3015
2500	0.2936

Также на рисунках 4.3–4.4 приведены графические результаты замеров.

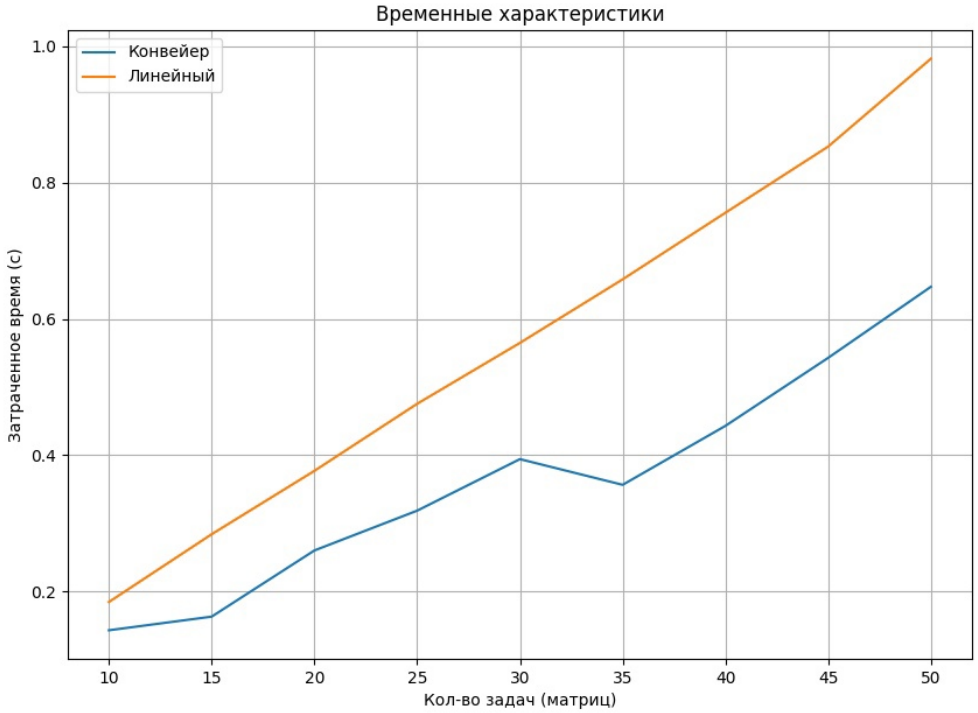

Рисунок 4.3 – Сравнение по времени линейной и конвейерной обработок для разного кол-ва матриц

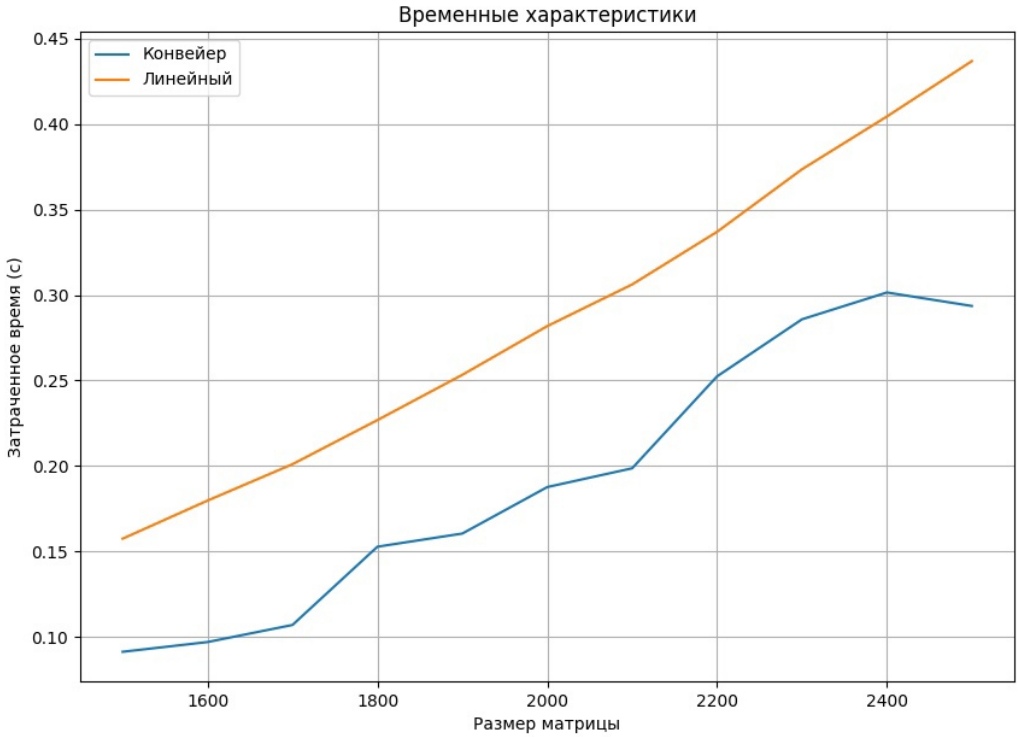

Рисунок 4.4 – Сравнение по времени линейной и конвейерной обработок для разных размеров матриц

4.4 Вывод

В результате эксперимента было получено, что использование конвейрной обработки лучше линейной реализации при количестве матриц, равном 10, в 1.3 раза, а при количестве матриц, котрое равно 50, уже в 1.5 раза. Следовательно, конвейерная реализация лучше линейной при увеличении количества задач (матриц).

Также при проведении эксперимента было выявлено, что при увеличении размера матриц конвейерная реализация выдает лучшие результаты. Так, при размере матрицы в 1500 конвейерная реализация лучше в 1.4 раза, чем линейная, а при размере матриц, равном 2500, в 1.5 раза. То есть, следует использоовть конвейрную реализацию.

Заключение

В результате исследования было определено, что конвейерная реализация лучше линейной при изменении обоих критериев – количества задач (матриц) и при увеличении размеров матриц. Так, использование конвейрного подхода дает прирост в 1.3 раза при 10 матрицах и 1.5 раза при 50 матрицах, а при размере матриц в 1500 время лучше в 1.4 раза, а при 2500 – в 1.5 раз.

Цель, которая была поставлена в начале лабораторной работы была достигнута, а также в ходе выполнения лабораторной работы были решены следующие задачи:

- были изучены принципы конвейерной разработки;
- были реализованы алгоритмы обрбаотки матрицы – нахождение среднего арифметического, нахождение максимального элемента и заполнение матрицы найденным значениями;
- проведен сравнительный анализ по времени алгоритмов конвейрной и линейной обработок на разном количестве задач (матриц);
- проведен сравнительный анализ по времени алгоритмов конвейрной и линейной обработок на разном размере матриц;
- подготовлен отчет о лабораторной работе.

Список литературы

[1] Конвейерная обработка данных [Электронный ресурс]. Режим доступа: https://studref.com/636041/ekonomika/konveyernaya_obrabotka_dannyh (дата обращения: 23.10.2021).

[2] Программирование на C/C++ [Электронный ресурс]. Режим доступа: http://www.c-cpp.ru/ (дата обращения: 23.10.2021).

[3] Welcome to Python [Электронный ресурс]. Режим доступа: https://www.python.org (дата обращения: 23.10.2021).

[4] Date and time utilities [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/cpp/chrono (дата обращения: 23.10.2021).

[5] Ubuntu 20.04.3 LTS (Focal Fossa) [Электронный ресурс]. Режим доступа: https://releases.ubuntu.com/20.04/ (дата обращения: 15.10.2021).

[6] Linux [Электронный ресурс]. Режим доступа: https://www.linux.org/forums/#linux-tutorials.122 (дата обращения: 15.10.2021).

[7] Процессор Intel® Core™ i5-7300HQ [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/97456/intel-core-i5-7300hq-processor-6m-cache-up-to-3-50-ghz.html (дата обращения: 04.10.2021).