



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»

Отчёт по лабораторной работе № 5 по курсу "Анализ алгоритмов"

Тема Конвейерная обработка данных

Студент Калашков П. А.

Группа ИУ7-56Б

Оценка (баллы)

Преподаватели Волкова Л. Л., Строганов Ю. В.

Содержание

| | |
|--|-----------|
| Введение | 3 |
| 1 Аналитическая часть | 4 |
| 1.1 Конвейерная обработка данных | 4 |
| 1.2 Описание алгоритмов | 4 |
| 2 Конструкторская часть | 6 |
| 2.1 Описание используемых типов данных | 6 |
| 2.2 Разработка алгоритмов | 6 |
| 2.3 Классы эквивалентности при функциональном тестировании . . | 12 |
| 3 Технологическая часть | 13 |
| 3.1 Средства реализации | 13 |
| 3.2 Сведения о модулях программы | 13 |
| 3.3 Реализации алгоритмов | 13 |
| 3.4 Функциональные тесты | 18 |
| 4 Исследовательская часть | 19 |
| 4.1 Технические характеристики | 19 |
| 4.2 Демонстрация работы программы | 19 |
| 4.3 Время выполнения реализаций алгоритмов | 22 |
| Заключение | 27 |
| Список использованных источников | 28 |

Введение

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

Цель работы: изучение принципов конвейерной обработки данных.

Задачи работы:

- 1) изучить основы конвейерной обработки данных;
- 2) описать алгоритмы обработки матрицы, которые будут использоваться в текущей лабораторной работе;
- 3) сравнить и проанализировать реализации алгоритмов по затраченному времени;
- 4) описать и обосновать полученные результаты в отчёте о выполненной лабораторной работе, выполненном как расчётно-пояснительная записка к работе.

1 Аналитическая часть

В этом разделе будет рассмотрена информация, касающаяся основ конвейерной обработки данных.

1.1 Конвейерная обработка данных

Конвейер [1] (англ. *conway*) — 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 позволяет увеличить 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 операции, подвергающейся конвейерной обработке, будет обрабатываться матрица. Всего будет использовано три ленты, ко-

которые делают следующее:

- 1) находится среднее арифметическое значений матрицы;
- 2) находится максимальный элемент матрицы;
- 3) каждый нечетный элемент матрицы заменяется на среднее арифметическое матрицы, а четные — на максимальный элемент.

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 программное обеспечение будет давать возможность получить журнал программы для установленного числа задач при линейной и конвейерной обработке. Также будет возможность провести тестирование по времени для разного количества задач (матриц) и разных размеров самих матриц.

2 Конструкторская часть

В этом разделе будут рассмотрены описания используемых типов данных, а также схемы алгоритмов конвейерной и линейной обработки матриц.

2.1 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие типы данных:

- количество задач (матриц) — целое число;
- размер матрицы — целое число;
- структура *matrix_s* — содержит информацию о матрице — элементы матрицы, её размер, а также информацию о найденном среднем арифметическом и максимальном элементах;
- структура *queues_t* — содержит информацию об очередях.

2.2 Разработка алгоритмов

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

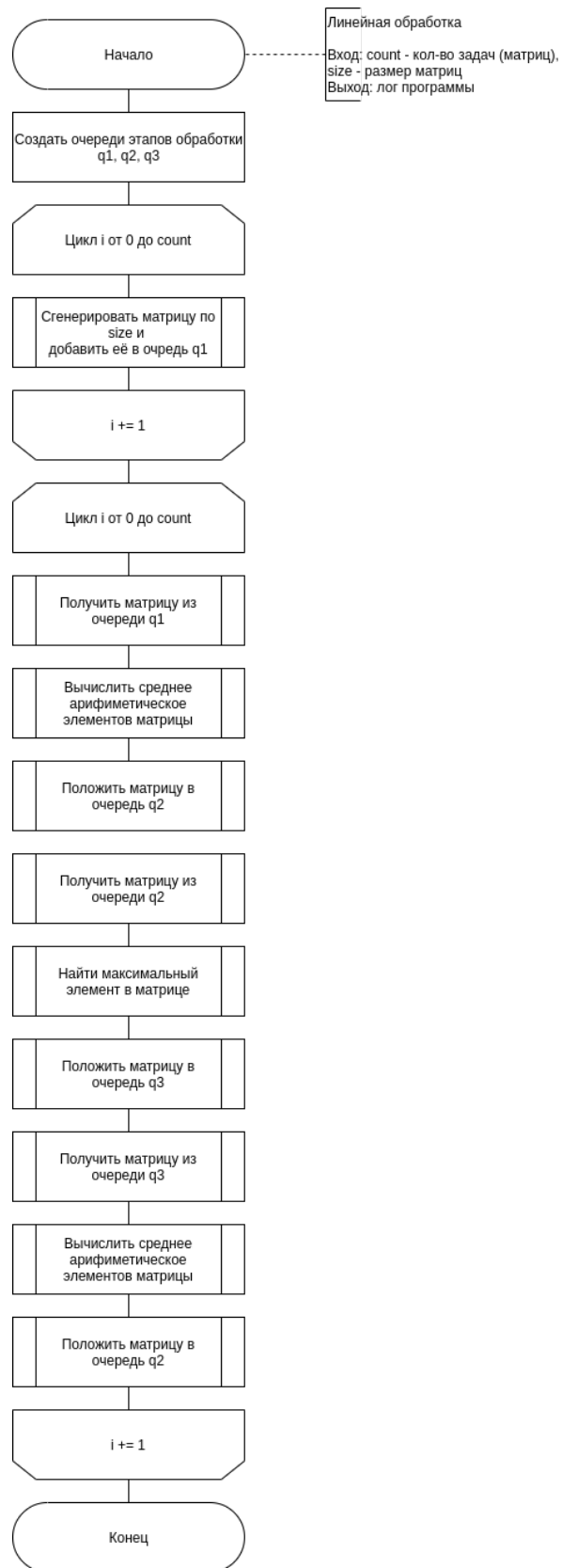


Рисунок 2.1 – Схема алгоритма линейной обработки матрицы

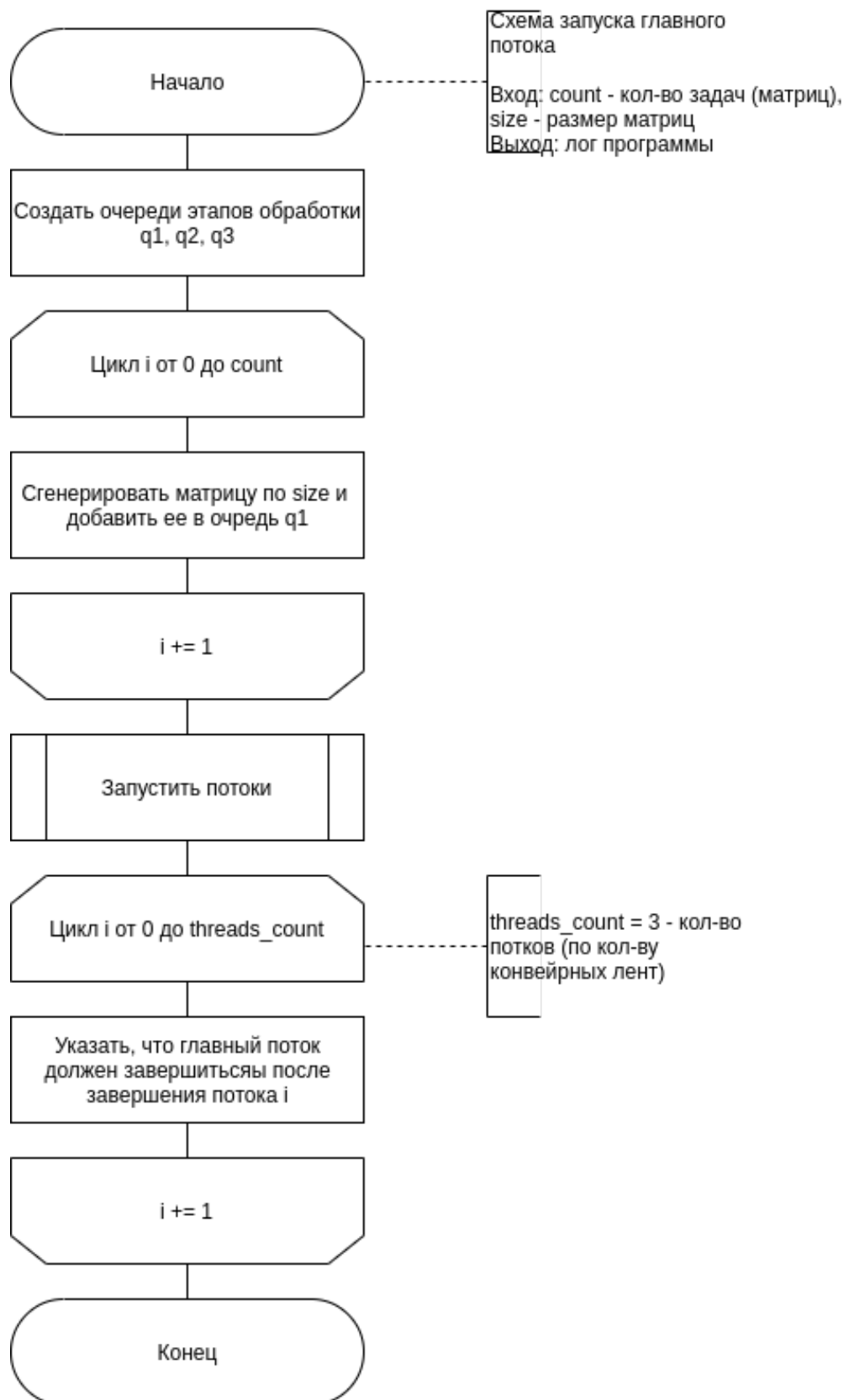


Рисунок 2.2 – Схема конвейерной обработки матрицы

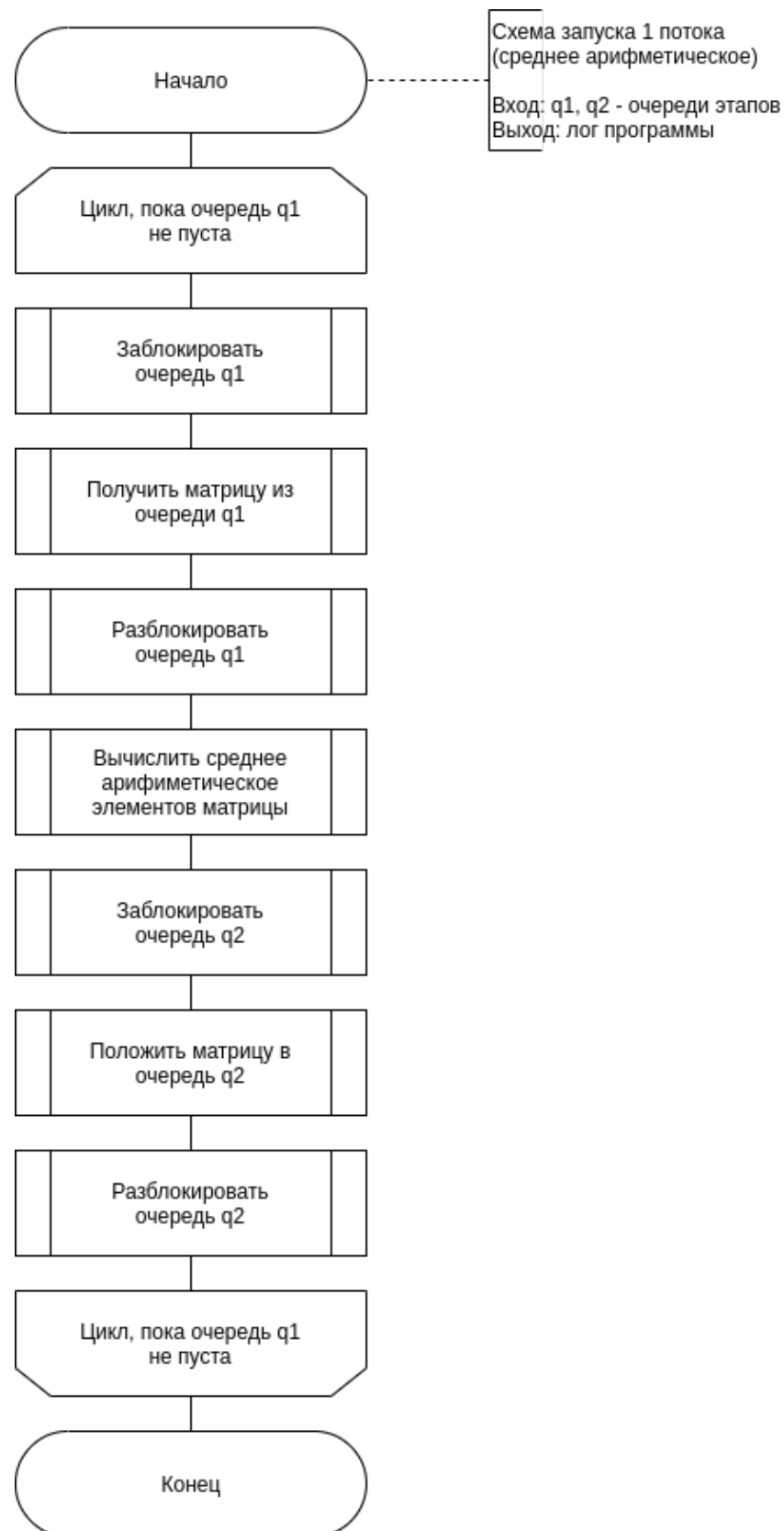


Рисунок 2.3 – Схема 1 потока обработки матрицы — нахождение среднего арифметического

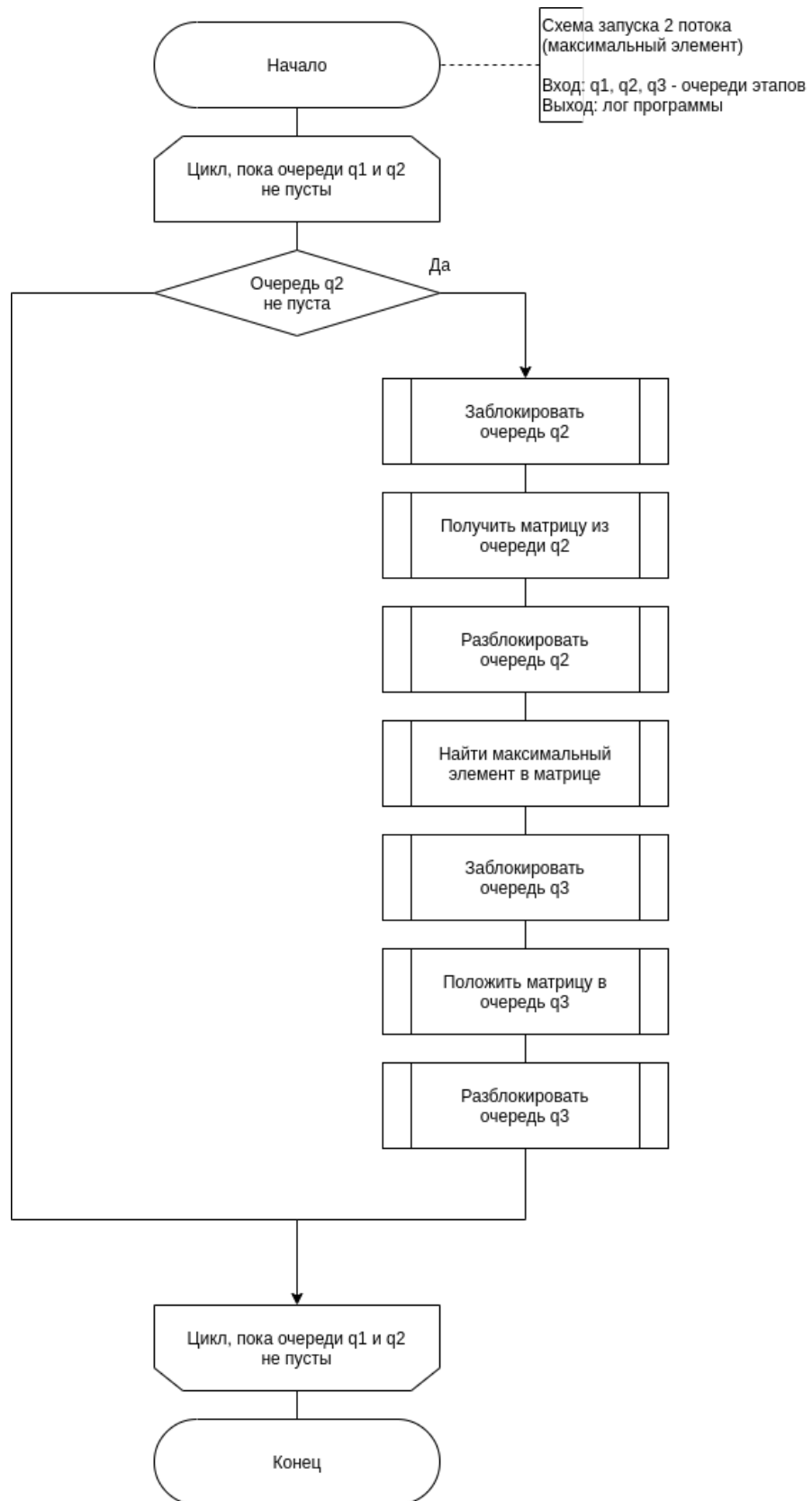


Рисунок 2.4 – Схема 2 потока обработки матрицы — нахождение максимального элемента

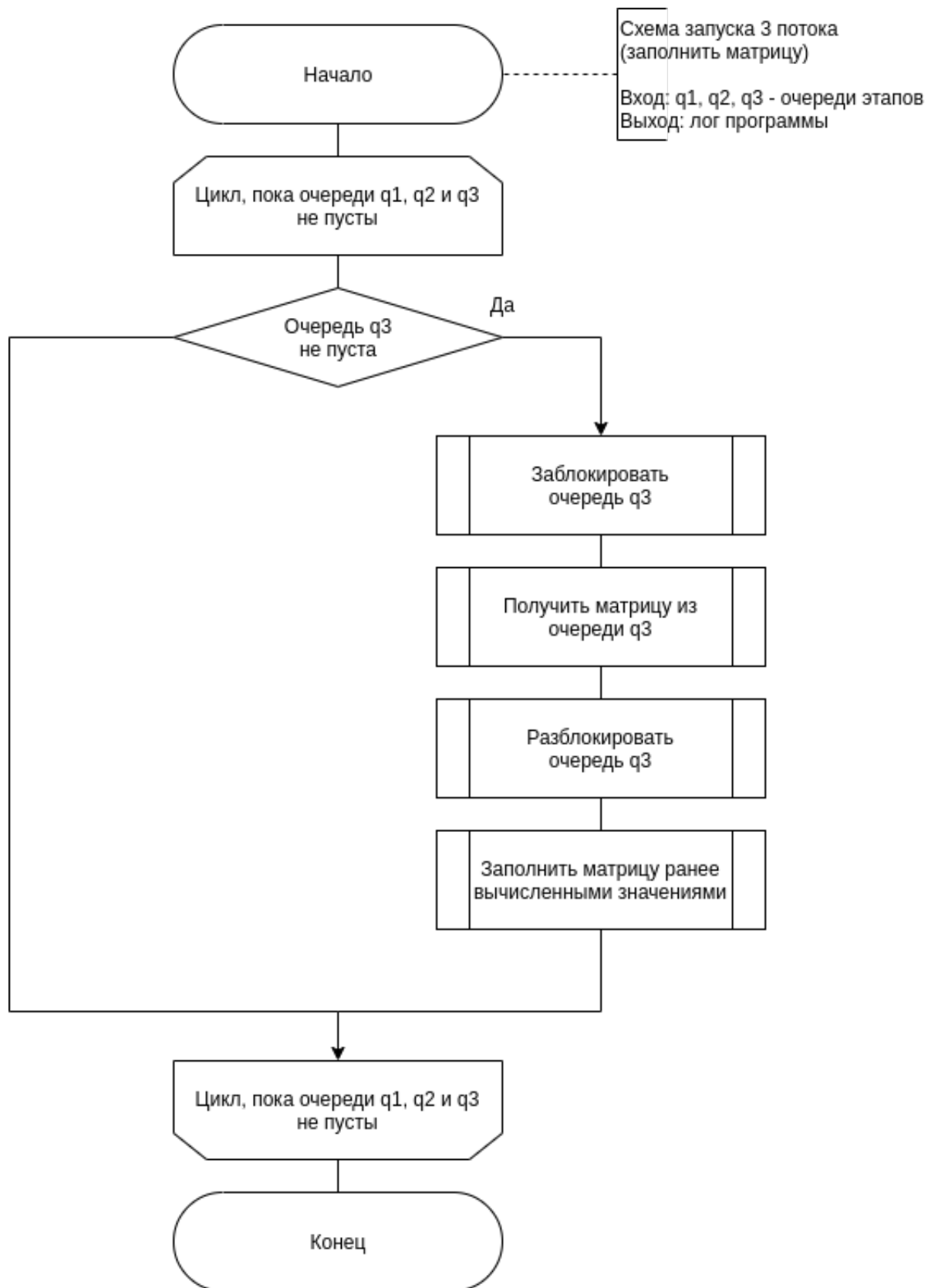


Рисунок 2.5 – Схема 3 потока обработки матрицы — заполнения матрицы новыми значениями

2.3 Классы эквивалентности при функциональном тестировании

Для функционального 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.

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 конвейерной обработки матрицы.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования *C++* [2]. В текущей лабораторной работе требуется замерить время работы выполняемой программы и визуализировать результаты при помощи графиков. Инструменты для этого присутствуют в выбранном языке программирования.

Построение графиков и функциональное тестирование было осуществлено при помощи языка программирования *Python* [3].

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

3.2 Сведения о модулях программы

Программа состоит из пяти модулей:

- 1) *main.cpp* — файл, содержащий точку входа;
- 2) *conway* — модуль, содержащий код линейной и конвейерной обработок;
- 3) *matrix* — модуль, содержащий код функций обработки матрицы;
- 4) *menu* — модуль, содержащий алгоритм меню;
- 5) *utils* — модуль, содержащий служебные алгоритмы.

3.3 Реализации алгоритмов

В листингах 3.1–3.6 представлены реализации алгоритмов линейной и конвейерной обработок матрицы, а также алгоритмы запуска трёх потоков (для

нахождения среднего арифметического значений матрицы, для нахождения максимального элемента и для заполнения матрицы соответственно).

Листинг 3.1 – Алгоритм линейной обработки матрицы

```
1  void parseLinear(int count, size_t size, bool needPrinting) {
2      currentTime = 0;
3      std::queue<matrix_t> q1;
4      std::queue<matrix_t> q2;
5      std::queue<matrix_t> q3;
6
7      queues_t queues = {.q1 = q1, .q2 = q2, .q3 = q3};
8
9      for (int i = 0; i < count; i++) {
10         matrix_t res = generateMatrix(size);
11
12         queues.q1.push(res);
13     }
14
15     for (int i = 0; i < count; i++) {
16         matrix_t matrix = queues.q1.front();
17         stage1Linear(matrix, i + 1, needPrinting);
18         queues.q1.pop();
19         queues.q2.push(matrix);
20
21         matrix = queues.q2.front();
22         stage2Linear(matrix, i + 1, needPrinting);
23         queues.q2.pop();
24         queues.q3.push(matrix);
25
26         matrix = queues.q3.front();
27         stage3Linear(matrix, i + 1, needPrinting);
28         queues.q3.pop();
29     }
30 }
```

Листинг 3.2 – Алгоритм конвейерной обработки матрицы

```
1 void parseParallel(int count, size_t size, bool needPrinting) {
2     t1.resize(count + 1);
3     t2.resize(count + 1);
4     t3.resize(count + 1);
5
6     for (int i = 0; i < count + 1; i++) {
7         t1[i] = 0;
8         t2[i] = 0;
9         t3[i] = 0;
10    }
11
12    std::queue<matrix_t> q1;
13    std::queue<matrix_t> q2;
14    std::queue<matrix_t> q3;
15
16    queues_t queues = {.q1 = q1, .q2 = q2, .q3 = q3};
17
18
19    for (int i = 0; i < count; i++) {
20        matrix_t res = generateMatrix(size);
21        q1.push(res);
22    }
23
24    std::thread threads[THREADS];
25
26    threads[0] = std::thread(stage1Parallel, std::ref(q1),
27        std::ref(q2), std::ref(q3), needPrinting);
28    threads[1] = std::thread(stage2Parallel, std::ref(q1),
29        std::ref(q2), std::ref(q3), needPrinting);
30    threads[2] = std::thread(stage3Parallel, std::ref(q1),
31        std::ref(q2), std::ref(q3), needPrinting);
32
33    for (int i = 0; i < THREADS; i++) {
34        threads[i].join();
35    }
36 }
```

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

```
1 void stage1Parallel(std::queue<matrix_t> &q1, std::queue<matrix_t>
  &q2, std::queue<matrix_t> &q3, bool needPrinting) {
2
3     int task = 1;
4     std::mutex m;
5     while(!q1.empty()) {
6         m.lock();
7         matrix_t matrix = q1.front();
8         m.unlock();
9
10        logConway(matrix, task++, 1, getAverage, needPrinting);
11
12        m.lock();
13        q2.push(matrix);
14        q1.pop();
15        m.unlock();
16    }
17 }
```

Листинг 3.4 – Алгоритм запуска 2 потока для нахождения максимального элемента матрицы (часть 1)

```
1 void stage2Parallel(std::queue<matrix_t> &q1, std::queue<matrix_t>
  &q2, std::queue<matrix_t> &q3, bool needPrinting) {
2
3     int task = 1;
4     std::mutex m;
5     do {
6         m.lock();
7         bool is_q2empty = q2.empty();
8         m.unlock();
9
10        if (!is_q2empty) {
11            m.lock();
12            matrix_t matrix = q2.front();
13            m.unlock();
14
15            logConway(matrix, task++, 2, getMax, needPrinting);
```


Листинг 3.5 – Алгоритм запуска 2 потока для нахождения максимального элемента матрицы (часть 2)

```
1         m.lock();
2         q3.push(matrix);
3         q2.pop();
4         m.unlock();
5     }
6 } while (!q1.empty() || !q2.empty());
7 }
```

Листинг 3.6 – Алгоритм запуска 3 потока для заполнения матрицы
вычисленными значениями

```
1 void stage3Parallel(std::queue<matrix_t> &q1, std::queue<matrix_t>
   &q2, std::queue<matrix_t> &q3, bool needPrinting) {
2
3     int task = 1;
4     std::mutex m;
5
6     do {
7         m.lock();
8         bool is_q3empty = q3.empty();
9         m.unlock();
10
11         if (!is_q3empty) {
12             m.lock();
13             matrix_t matrix = q3.front();
14             m.unlock();
15
16             logConway(matrix, task++, 3, fillMatrix, needPrinting);
17
18             m.lock();
19             q3.pop();
20             m.unlock();
21         }
22     } while (!q1.empty() || !q2.empty() || !q3.empty());
23 }
```

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 |
|-------------|---------------|---------------|-----------------------------|
| Конвейерная | -5 | 500 | Сообщение об ошибке |
| Конвейерная | 10 | -23 | Сообщение об ошибке |
| Линейная | 50 | 1500 | Вывод лога работы программы |
| Конвейерная | 10 | 100 | Вывод лога работы программы |
| Конвейерная | 5 | 10 | Вывод лога работы программы |

Вывод

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

4 Исследовательская часть

В данном разделе будут приведён пример работы программы, а также проведён сравнительный анализ алгоритмов при различных ситуациях на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени работы реализации алгоритма статической раздачи информации, представлены далее:

- операционная система Mac OS Monterey Версия 12.5.1 (21G83) [5] x86_64;
- память 16 ГБ;
- четырёхъядерный процессор Intel Core i7 с тактовой частотой 2,7 ГГц [6].

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 рисунках 4.1, 4.2 представлены результаты работы программы: замеры времени линейной и конвейерной обработки соответственно.

```
○ p.kalashkov in ~/Desktop/fifthTerm/bmstu-aa/lab05/app/build on branch master > ./app
```

Ступенчатая обработка матрицы

1. Линейная обработка
 2. Конвейерная обработка
 3. Замерить время
 4. Вывести информацию об этапах обработки
 0. Выход
- Выбор: 1

Размер: 100

Количество: 10

| | | | | | | | |
|-------|-----|--------|----|--------|-----------|------|----------|
| Task: | 1, | Stage: | 1, | Start: | 0.000000, | End: | 0.000053 |
| Task: | 1, | Stage: | 2, | Start: | 0.000053, | End: | 0.000128 |
| Task: | 1, | Stage: | 3, | Start: | 0.000128, | End: | 0.000191 |
| Task: | 2, | Stage: | 1, | Start: | 0.000191, | End: | 0.000244 |
| Task: | 2, | Stage: | 2, | Start: | 0.000244, | End: | 0.000319 |
| Task: | 2, | Stage: | 3, | Start: | 0.000319, | End: | 0.000381 |
| Task: | 3, | Stage: | 1, | Start: | 0.000381, | End: | 0.000436 |
| Task: | 3, | Stage: | 2, | Start: | 0.000436, | End: | 0.000510 |
| Task: | 3, | Stage: | 3, | Start: | 0.000510, | End: | 0.000569 |
| Task: | 4, | Stage: | 1, | Start: | 0.000569, | End: | 0.000619 |
| Task: | 4, | Stage: | 2, | Start: | 0.000619, | End: | 0.000688 |
| Task: | 4, | Stage: | 3, | Start: | 0.000688, | End: | 0.000747 |
| Task: | 5, | Stage: | 1, | Start: | 0.000747, | End: | 0.000797 |
| Task: | 5, | Stage: | 2, | Start: | 0.000797, | End: | 0.000868 |
| Task: | 5, | Stage: | 3, | Start: | 0.000868, | End: | 0.000927 |
| Task: | 6, | Stage: | 1, | Start: | 0.000927, | End: | 0.000977 |
| Task: | 6, | Stage: | 2, | Start: | 0.000977, | End: | 0.001047 |
| Task: | 6, | Stage: | 3, | Start: | 0.001047, | End: | 0.001106 |
| Task: | 7, | Stage: | 1, | Start: | 0.001106, | End: | 0.001156 |
| Task: | 7, | Stage: | 2, | Start: | 0.001156, | End: | 0.001225 |
| Task: | 7, | Stage: | 3, | Start: | 0.001225, | End: | 0.001285 |
| Task: | 8, | Stage: | 1, | Start: | 0.001285, | End: | 0.001346 |
| Task: | 8, | Stage: | 2, | Start: | 0.001346, | End: | 0.001433 |
| Task: | 8, | Stage: | 3, | Start: | 0.001433, | End: | 0.001492 |
| Task: | 9, | Stage: | 1, | Start: | 0.001492, | End: | 0.001556 |
| Task: | 9, | Stage: | 2, | Start: | 0.001556, | End: | 0.001636 |
| Task: | 9, | Stage: | 3, | Start: | 0.001636, | End: | 0.001699 |
| Task: | 10, | Stage: | 1, | Start: | 0.001699, | End: | 0.001750 |
| Task: | 10, | Stage: | 2, | Start: | 0.001750, | End: | 0.001820 |
| Task: | 10, | Stage: | 3, | Start: | 0.001820, | End: | 0.001878 |

Рисунок 4.1 – Пример работы программы: линейная обработка матриц (размер матриц 100, количество 10).

○ p.kalashkov in ~/Desktop/fifthTerm/bmstu-aa/lab05/app/build on branch master > ./app

Ступенчатая обработка матрицы

1. Линейная обработка
 2. Конвейерная обработка
 3. Замерить время
 4. Вывести информацию об этапах обработки
 0. Выход
- Выбор: 2

Размер: 100

Количество: 10

| | | | | | | | |
|-------|-----|--------|----|--------|-----------|------|----------|
| Task: | 1, | Stage: | 1, | Start: | 0.000000, | End: | 0.000054 |
| Task: | 1, | Stage: | 2, | Start: | 0.000054, | End: | 0.000129 |
| Task: | 2, | Stage: | 1, | Start: | 0.000054, | End: | 0.000107 |
| Task: | 1, | Stage: | 3, | Start: | 0.000129, | End: | 0.000192 |
| Task: | 2, | Stage: | 2, | Start: | 0.000107, | End: | 0.000182 |
| Task: | 3, | Stage: | 1, | Start: | 0.000107, | End: | 0.000161 |
| Task: | 2, | Stage: | 3, | Start: | 0.000182, | End: | 0.000242 |
| Task: | 3, | Stage: | 2, | Start: | 0.000161, | End: | 0.000232 |
| Task: | 4, | Stage: | 1, | Start: | 0.000161, | End: | 0.000213 |
| Task: | 3, | Stage: | 3, | Start: | 0.000232, | End: | 0.000292 |
| Task: | 4, | Stage: | 2, | Start: | 0.000213, | End: | 0.000284 |
| Task: | 5, | Stage: | 1, | Start: | 0.000213, | End: | 0.000265 |
| Task: | 4, | Stage: | 3, | Start: | 0.000284, | End: | 0.000345 |
| Task: | 5, | Stage: | 2, | Start: | 0.000265, | End: | 0.000337 |
| Task: | 6, | Stage: | 1, | Start: | 0.000265, | End: | 0.000316 |
| Task: | 5, | Stage: | 3, | Start: | 0.000337, | End: | 0.000398 |
| Task: | 7, | Stage: | 1, | Start: | 0.000316, | End: | 0.000367 |
| Task: | 6, | Stage: | 2, | Start: | 0.000316, | End: | 0.000388 |
| Task: | 6, | Stage: | 3, | Start: | 0.000388, | End: | 0.000448 |
| Task: | 8, | Stage: | 1, | Start: | 0.000367, | End: | 0.000419 |
| Task: | 7, | Stage: | 2, | Start: | 0.000388, | End: | 0.000458 |
| Task: | 7, | Stage: | 3, | Start: | 0.000458, | End: | 0.000518 |
| Task: | 9, | Stage: | 1, | Start: | 0.000419, | End: | 0.000470 |
| Task: | 8, | Stage: | 2, | Start: | 0.000458, | End: | 0.000543 |
| Task: | 8, | Stage: | 3, | Start: | 0.000543, | End: | 0.000601 |
| Task: | 10, | Stage: | 1, | Start: | 0.000470, | End: | 0.000520 |
| Task: | 9, | Stage: | 2, | Start: | 0.000543, | End: | 0.000611 |
| Task: | 9, | Stage: | 3, | Start: | 0.000611, | End: | 0.000670 |
| Task: | 10, | Stage: | 2, | Start: | 0.000611, | End: | 0.000680 |
| Task: | 10, | Stage: | 3, | Start: | 0.000680, | End: | 0.000738 |

Рисунок 4.2 – Пример работы программы: конвейерная обработка матриц (размер матриц 100, количество 10).

4.3 Время выполнения реализаций 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 разных размеров 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 (линейный алгоритм — разное число матриц)

| Количество 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 (конвейерный алгоритм — разное число матриц)

| Количество 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 (линейный алгоритм — разные размеры матриц)

| Линейный 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 (конвейерный алгоритм – разные размеры матриц)

| Линейный размер матрицы | 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 результаты замеров.

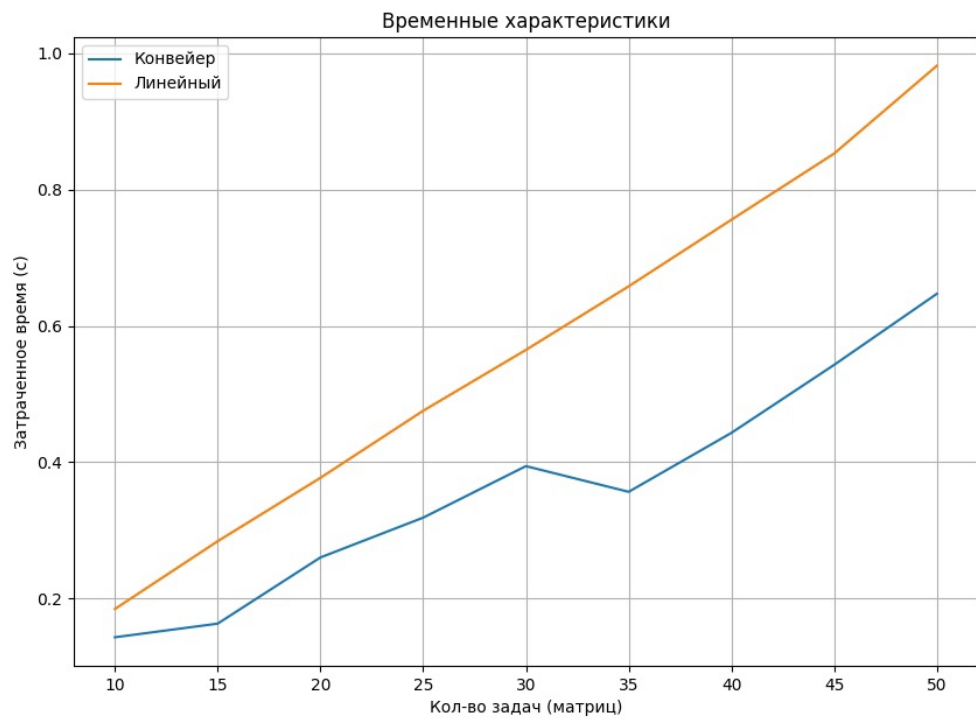


Рисунок 4.3 – Сравнение по времени линейной и конвейерной обработки для разного числа матриц

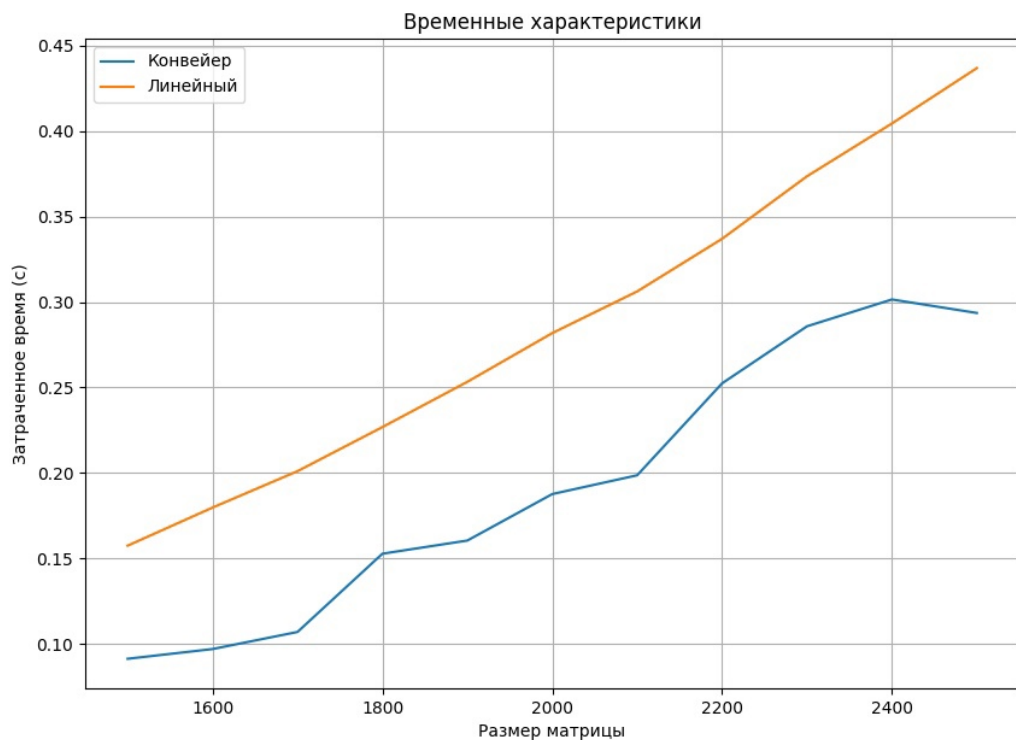


Рисунок 4.4 – Сравнение по времени линейной и конвейерной обработки для линейных размеров матриц

Вывод

В результате эксперимента было получено, что использование конвейерной обработки лучше (по времени выполнения) линейной реализации при количестве матриц, равном 10, в 1.3 раза, а при числе матриц, которое равно 50, уже в 1.5 раза. Следовательно, конвейерная реализация лучше линейной при увеличении количества задач (матриц) по этому же критерию.

Также при проведении эксперимента было выявлено, что при увеличении размера матриц конвейерная реализация выдает лучшие результаты. Так, при линейном размере матрицы в 1500 конвейерная реализация лучше в 1.4 раза, чем линейная, а при линейном размере матриц, равном 2500, в 1.5 раза. Таким образом, следует использовать конвейерную реализацию.

Заключение

Была достигнута цель работы: изучены принципы конвейерной обработки данных. Также в ходе выполнения лабораторной работы были решены следующие задачи:

- 1) были изучены основы конвейерной обработки данных;
- 2) были описаны используемые в лабораторной работе алгоритмы обработки матрицы;
- 3) были проведены сравнение и анализ трудоёмкостей алгоритмов на основе теоретических расчетов;
- 4) был подготовлен отчёт о лабораторной работе, представленный как расчётно-пояснительная записка к работе.

Исходя из полученных результатов, использование конвейерной обработки лучше (по времени выполнения) линейной реализации при количестве матриц, равном 10, в 1.3 раза, а при числе матриц, которое равно 50, уже в 1.5 раза. Следовательно, конвейерная реализация лучше линейной при увеличении количества задач (матриц) по этому же критерию.

Также при проведении эксперимента было выявлено, что при увеличении размера матриц конвейерная реализация выдает лучшие результаты. Так, при линейном размере матрицы в 1500 конвейерная реализация лучше в 1.4 раза, чем линейная, а при линейном размере матриц, равном 2500, в 1.5 раза. Таким образом, следует использовать конвейерную реализацию.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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. macOS Monterey [Электронный ресурс]. Режим доступа: <https://www.apple.com/macos/monterey/> (дата обращения: 17.09.2022).
6. Процессор Intel® Core™ i7 [Электронный ресурс]. Режим доступа: <https://www.intel.com/processors/core/i7/docs> (дата обращения: 17.09.2022).