



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 Организация асинхронного взаимодействия потоков вычисления

Студент Маслюков П.В.

Группа ИУ7-52Б

Оценка (баллы)

Преподаватель Волкова Л. Л.

Содержание

Введение	3
1 Аналитическая часть	4
1.1 Конвейерная обработка данных	4
2 Конструкторская часть	5
2.1 Разработка алгоритмов	5
3 Технологическая часть	11
3.1 Средства реализации	11
3.2 Реализация алгоритмов	11
3.3 Функциональные тесты	14
4 Исследовательская часть	15
4.1 Технические характеристики	15
4.2 Демонстрация работы программы	16
4.3 Время выполнения алгоритмов	17
Заключение	18
Список использованных источников	19

Введение

Задачу ускорения обработки данных можно решить с помощью введение конвейерной обработки. Вводится конвейерная лента и обрабатывающие устройства. Данные поступают на обрабатывающее устройство, которое после завершения обработки передает их дальше по ленте, и не ожидая завершения цикла, приступает к обработке следующих данных.

Целью работы является исследование конвейерной обработки данных.

Задачи, которые необходимо выполнить:

- 1) описать и реализовать алгоритм конвейерной обработки данных;
- 2) реализовать алгоритм линейной обработки данных;
- 3) провести тестирование по времени для этих алгоритмов;
- 4) провести сравнительный анализ по времени для этих алгоритмов.

1 Аналитическая часть

В этом разделе будет описан конвейерный принцип обработки данных.

1.1 Конвейерная обработка данных

Конвейер — 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[1]. Этот способ можно использовать в обработке данных, суть которой состоит в выделении отдельных этапов выполнения общей операции. Каждый этап, выполнив свою работу, передает результат следующему, одновременно принимая новую порцию данных.

2 Конструкторская часть

В этом разделе будут представлены схемы алгоритма линейной обработки данных и конвейерной обработки данных.

2.1 Разработка алгоритмов

На рис. 2.1-2.5 представлены схемы алгоритма линейной обработки данных, алгоритма конвейерной обработки данных и алгоритмов обрабатывающих устройств.

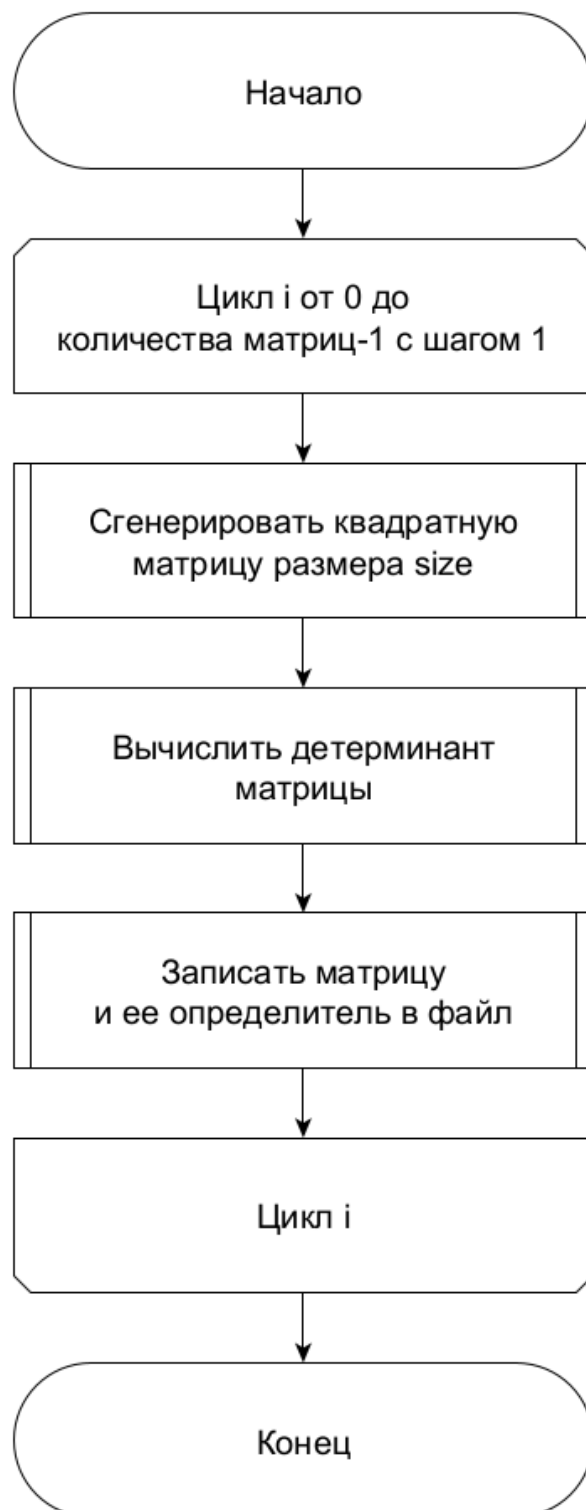


Рис. 2.1 – Алгоритм линейной обработки данных

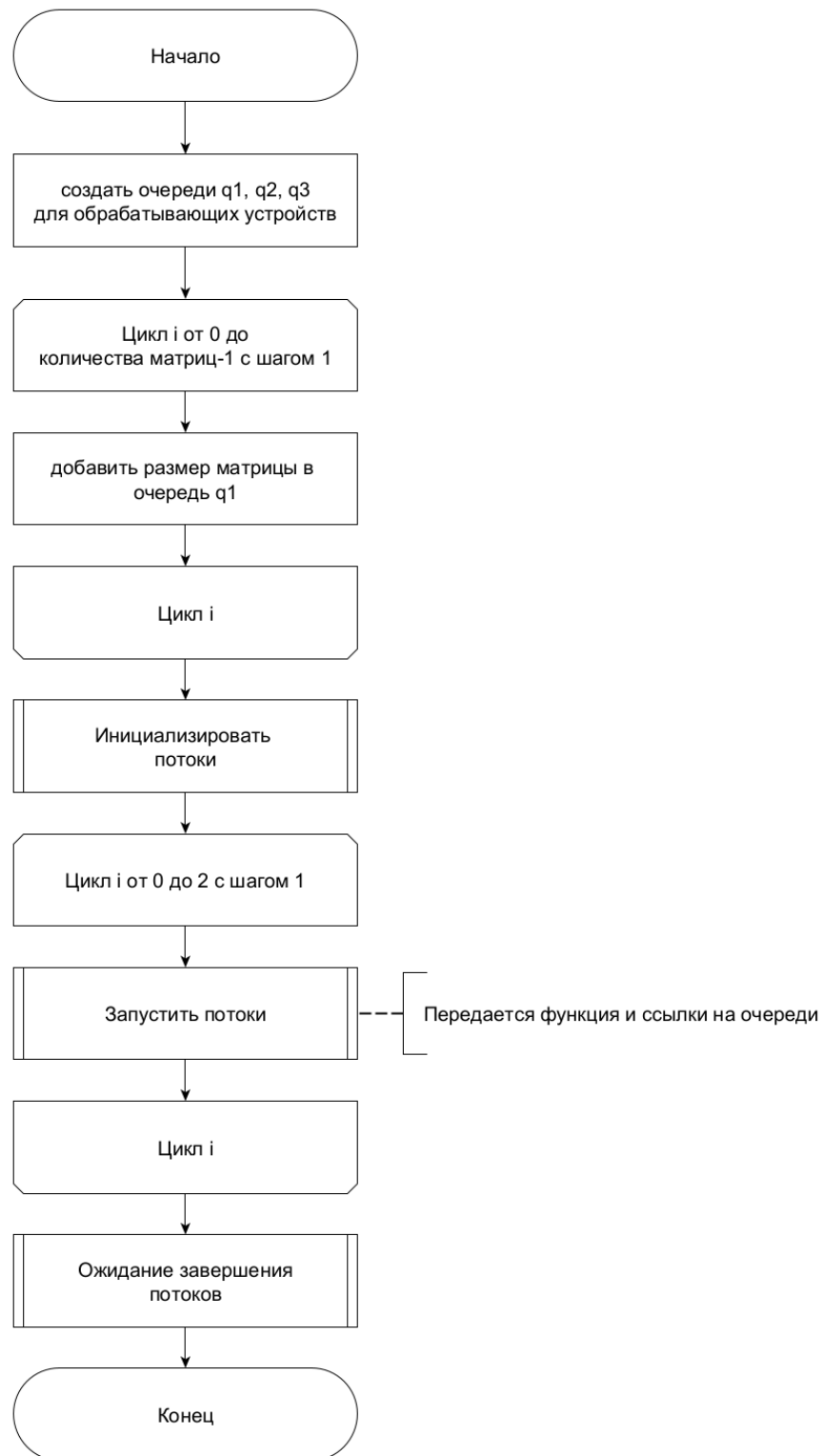


Рис. 2.2 – Алгоритм конвейерной обработки данных

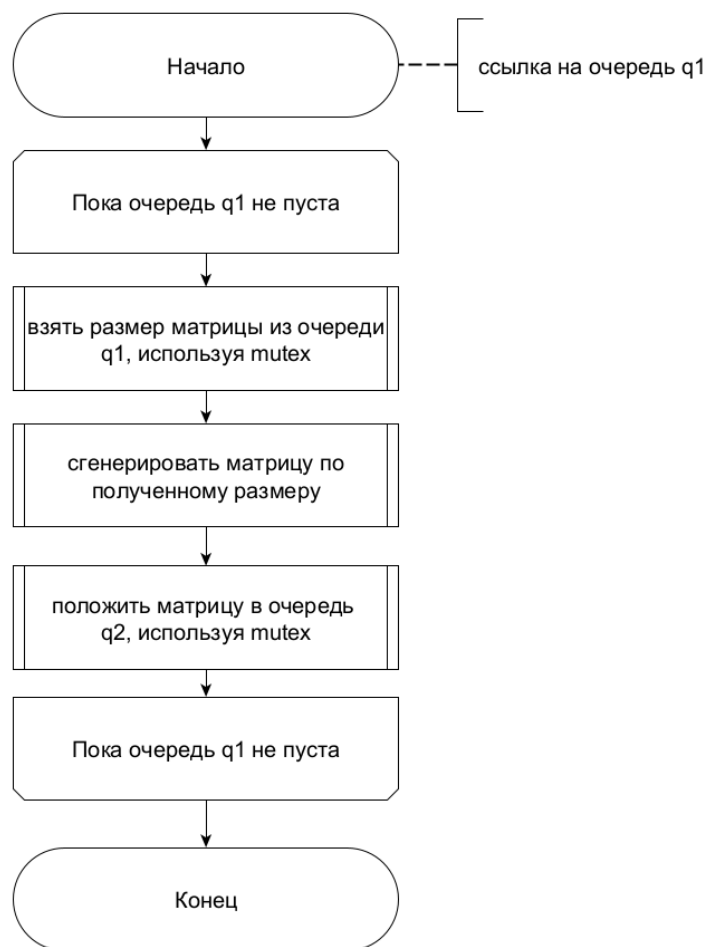


Рис. 2.3 – Алгоритм первого обрабатывающего устройства

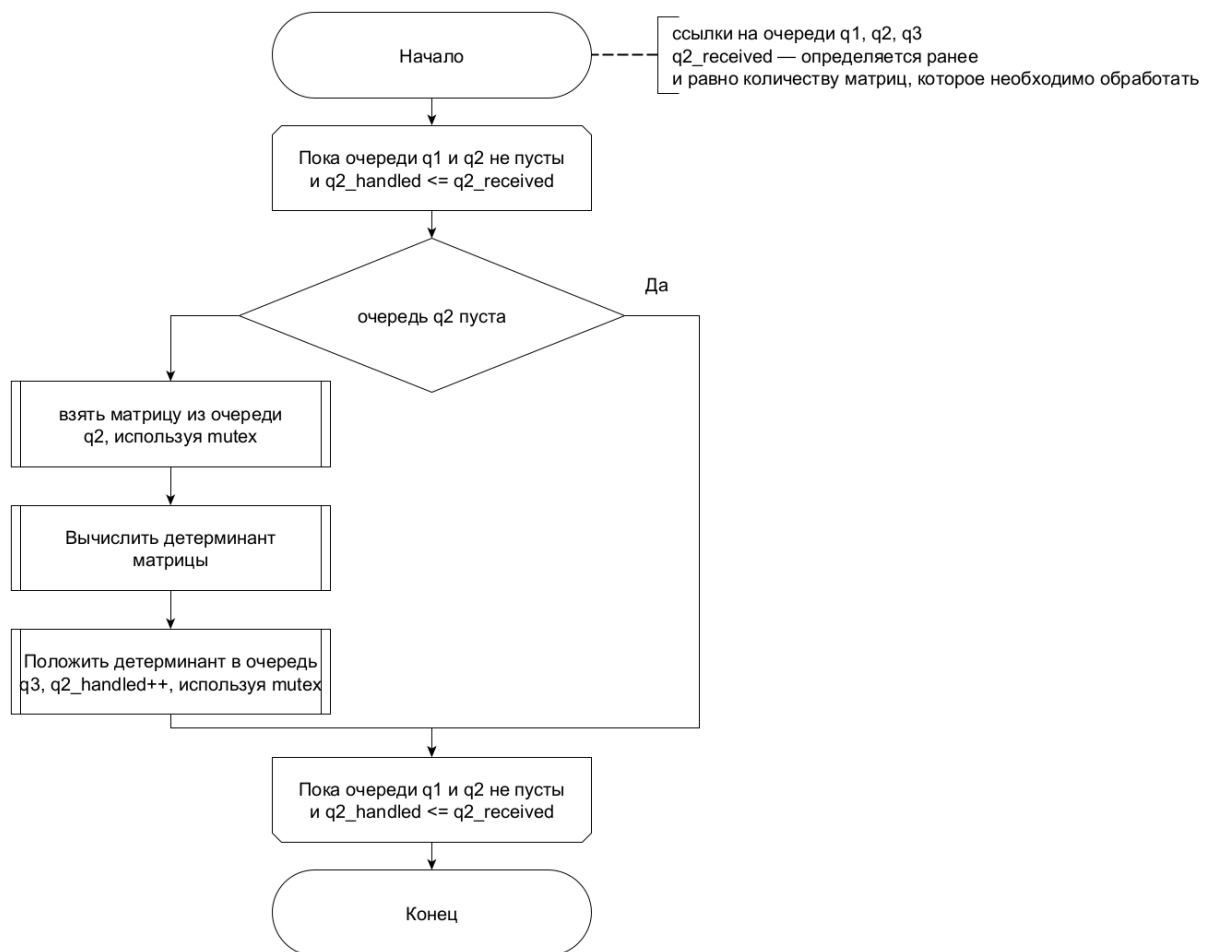


Рис. 2.4 – Алгоритм второго обрабатывающего устройства

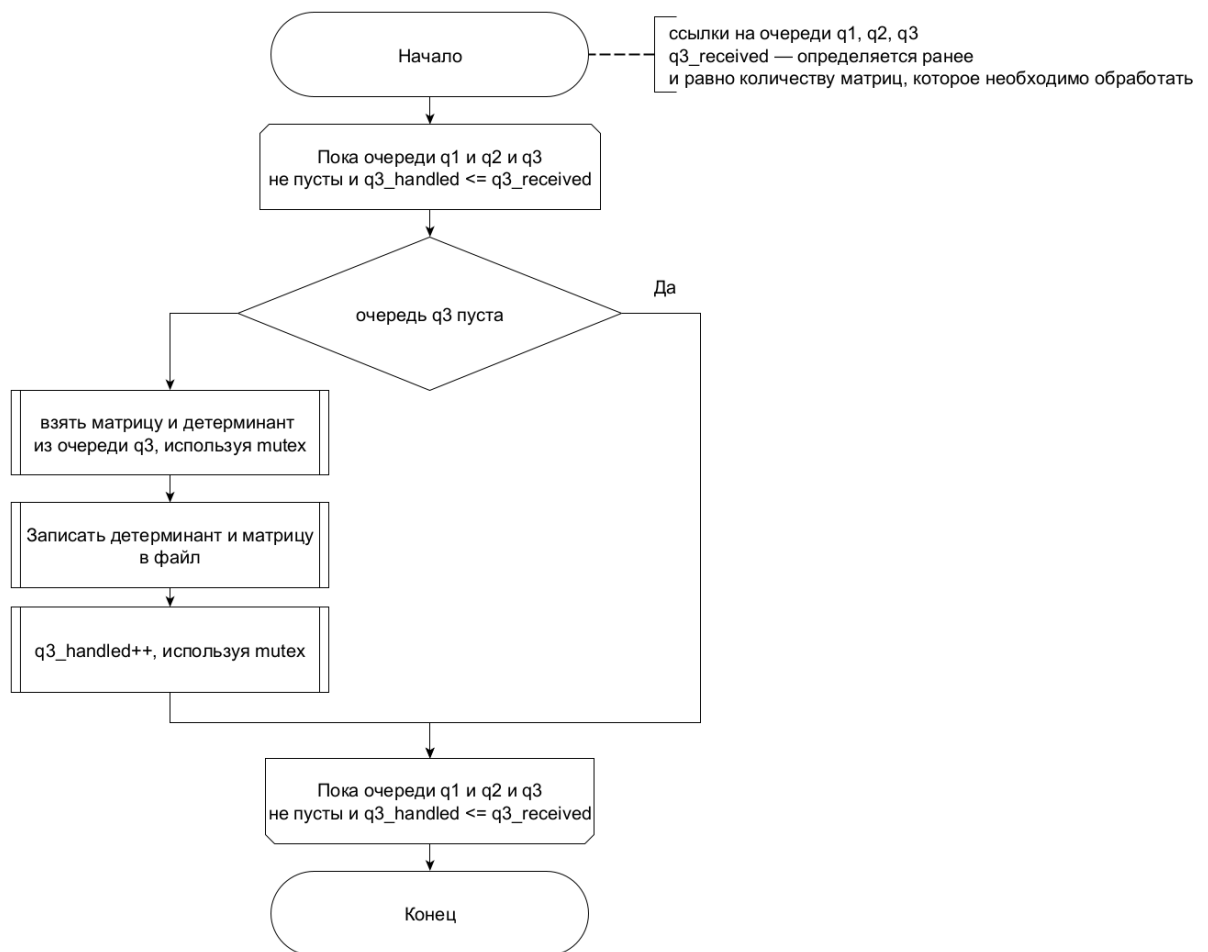


Рис. 2.5 – Алгоритм третьего обрабатывающего устройства

3 Технологическая часть

В данном разделе будут рассмотрены средства реализации, а также представлены листинги реализаций алгоритма линейной обработки данных и алгоритма конвейерной обработки данных.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования C++. Его возможностей достаточно для измерения процессорного времени и реализации алгоритмов.

Время работы было замерено с помощью функции `system_clock::now()`[2] из библиотеки `std::chrono`.

3.2 Реализация алгоритмов

В листингах 3.1-3.2 представлены реализации алгоритмов линейной и конвейерной обработки данных.

Листинг 3.1 – Алгоритм линейной обработки данных

```
1 void linear_execution(int count, size_t size, bool is_print)
2 {
3     time_now = 0;
4     for (int i = 0; i < count; i++)
5     {
6         matrix_t matrix = matrix_generate(size);
7         matrix.det = matrix_determinant(matrix, 0, matrix.size, 1);
8         matrix_dump(matrix);
9     }
10 }
11
```

Листинг 3.2 – Алгоритм конвейерной обработки данных

```
1 void parallel_execution(int count, size_t size)
2 {
3     std::queue<int> q1;
4     std::queue<matrix_t> q2;
5     std::queue<matrix_t> q3;
6     queues_t queues = {.q1 = q1, .q2 = q2, .q3 = q3};
7     for (int i = 0; i < count; i++)
8         q1.push(size);
9     std::thread threads[3];
10    threads[0] = std::thread(stage1_parallel, std::ref(q1), std::ref(q2), std
        ::ref(q3));
11    threads[1] = std::thread(stage2_parallel, std::ref(q1), std::ref(q2), std
        ::ref(q3));
12    threads[2] = std::thread(stage3_parallel, std::ref(q1), std::ref(q2), std
        ::ref(q3));
13    for (int i = 0; i < 3; i++)
14        threads[i].join();
15 }
```

Листинг 3.3 – Алгоритм генерации квадратной матрицы

```
1 matrix_t matrix_generate(size_t size)
2 {
3     std::vector<std::vector<int>>> tmp_data;
4     tmp_data.resize(size);
5     for (size_t i = 0; i < size; i++)
6         tmp_data[i].resize(size);
7     matrix_t matrix;
8     matrix.size = size;
9     matrix.data = tmp_data;
10    matrix.det = 0;
11    for (size_t i = 0; i < matrix.size; i++)
12        for (size_t j = 0; j < matrix.size; j++)
13            matrix.data[i][j] = std::experimental::randint(1, 10);
14    return matrix;
15 }
16
```

Листинг 3.4 – Алгоритм вычисления детерминанта матрицы

```
1 int matrix_determinant(matrix_t &matrix, int start, int end, int newDegree)
2 {
3     int det = 0;
4     int degree = newDegree;
5     int size = matrix.size;
6     if(size == 1)
7         return matrix.data[0][0];
8     else if(size == 2)
9         return matrix.data[0][0]*matrix.data[1][1] - matrix.data[0][1]*matrix.data
10            [1][0];
11     else
12     {
13         for(int j = start; j < end; j++)
14         {
15             matrix_t copy = matrix_copy(matrix);
16             matrix_WithoutRowAnColumn(copy, 0, j);
17             det += degree * matrix.data[0][j] * matrix_determinant(copy, 0, copy.
18                size, 1);
19             degree = -degree;
20         }
21     }
22     return det;
23 }
```

Листинг 3.5 – Алгоритм записи матрицы и детерминанта в файл

```

1 void matrix_dump(matrix_t matrix)
2 {
3     ofstream logf(LOG, ios::app);
4     if (logf.is_open())
5     {
6         for (size_t i = 0; i < matrix.size; i++)
7         {
8             for (size_t j = 0; j < matrix.size; j++)
9                 logf << matrix.data[i][j] << " ";
10            logf << "\n";
11        }
12        logf << "\n\nDet: " << matrix.det << "\n-----\n";
13        logf.close();
14    }
15 }

```

3.3 Функциональные тесты

В таблице 3.1 представлены функциональные тесты для программы.

Таблица 3.1 – Функциональные тесты

Матрица	Линейная	Конвейерная	Эталон
$\begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 3 \\ 2 & 4 & 0 \end{pmatrix}$	14	14	14
$\begin{pmatrix} 4 & 7 & 2 & 1 & 4 \\ 4 & 2 & 3 & 5 & 2 \\ 4 & 3 & 7 & 9 & 1 \\ 3 & 7 & 9 & 2 & 9 \\ 7 & 9 & 3 & 1 & 7 \end{pmatrix}$	-867	-867	-867
$\begin{pmatrix} 1 & 3 & 2 & 2 \\ 7 & 4 & 2 & 5 \\ 4 & 6 & 1 & 4 \\ 2 & 6 & 9 & 7 \end{pmatrix}$	71	71	71

4 Исследовательская часть

В данном разделе будет проведен сравнительный анализ алгоритмов по времени выполнения в зависимости от количества матриц и их размеров.

4.1 Технические характеристики

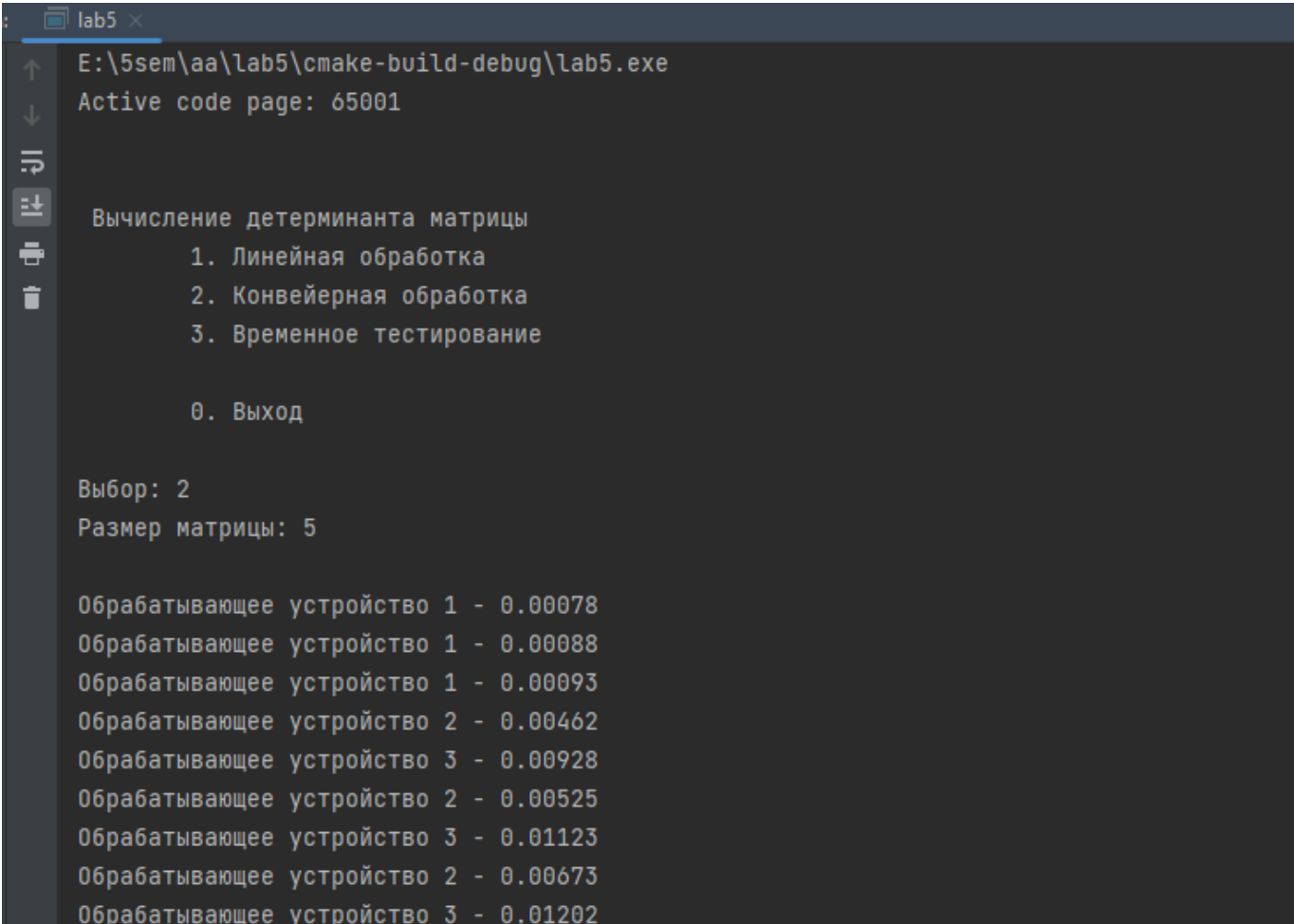
Технические характеристики устройства, на котором выполнялось тестирование представлены далее:

- 1) операционная система: Windows 10 pro;
- 2) память: 32 Гб;
- 3) процессор: 12h Gen Intel(R) Core(TM) i5-12400 2.50 ГГц.

Во время замеров компьютер был включен в сеть электропитания и был нагружен только системными программами и программой замеров.

4.2 Демонстрация работы программы

На рис. 4.1 представлена демонстрация работы программы.



```
lab5 x
E:\5sem\aa\lab5\cmake-build-debug\lab5.exe
Active code page: 65001

Вычисление детерминанта матрицы
    1. Линейная обработка
    2. Конвейерная обработка
    3. Временное тестирование

    0. Выход

Выбор: 2
Размер матрицы: 5

Обрабатывающее устройство 1 - 0.00078
Обрабатывающее устройство 1 - 0.00088
Обрабатывающее устройство 1 - 0.00093
Обрабатывающее устройство 2 - 0.00462
Обрабатывающее устройство 3 - 0.00928
Обрабатывающее устройство 2 - 0.00525
Обрабатывающее устройство 3 - 0.01123
Обрабатывающее устройство 2 - 0.00673
Обрабатывающее устройство 3 - 0.01202
```

Рис. 4.1 – Демонстрация работы программы

4.3 Время выполнения алгоритмов

Результаты замеров приведены в таблицах 4.1-4.4. Время указано в секундах.

Таблица 4.1 – Результаты замеров по времени для матриц размером 7

Количество матриц	Линейная	Конвейерная
1	0.0109	0.0118
2	0.0223	0.0231
3	0.0294	0.0331
4	0.0430	0.0378

Таблица 4.2 – Результаты замеров по времени для матриц размером 8

Количество матриц	Линейная	Конвейерная
1	0.0815	0.0495
2	0.1555	0.0940
3	0.2385	0.1402
4	0.3259	0.1944

Таблица 4.3 – Результаты замеров по времени для матриц размером 9

Количество матриц	Линейная	Конвейерная
1	0.7050	0.4097
2	1.4126	0.8191
3	2.1326	1.2208
4	2.8180	1.5875

Таблица 4.4 – Результаты замеров по времени для матриц размером 10

Количество матриц	Линейная	Конвейерная
1	7.0369	3.6058
2	14.1037	7.1952
3	21.2308	10.6358
4	28.1996	14.8571

Заключение

В результате было определено, что конвейерная обработка данных работает быстрее при размере матрицы равном 7 и количестве матриц от 4-х. Начиная с размера матрицы равного 8-ми, конвейерная обработка работает быстрее, чем линейная обработка.

Цель, лабораторной работы была достигнута, была исследована конвейерная обработка данных.

В ходе выполнения лабораторной работы были решены следующие задачи:

- 1) описан и реализован алгоритм конвейерной обработки данных;
- 2) реализован алгоритм линейной обработки данных;
- 3) проведено тестирование по времени для этих алгоритмов;
- 4) проведен сравнительный анализ по времени для этих алгоритмов.

Список использованных источников

- [1] Конвейерная обработка данных [Электронный ресурс]. Режим доступа: https://studref.com/636041/ekonomika/konveyernaya_obrabotka_dannyh (дата обращения: 06.01.2024).
- [2] `std::chrono::system_clock::now` [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/cpp/chrono/system_clock/now (дата обращения: 06.01.2024).