



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 Рубежному контролю  
по курсу «Анализ Алгоритмов»

Тема Многопоточная обработка данных

---

Студент Маслюков П.В.

---

Группа ИУ7-52Б

---

Оценка (баллы)

---

Преподаватель Волкова Л. Л.

---

Москва — 2024 г.

# Содержание

<b>Введение</b>	<b>3</b>
<b>1 Аналитическая часть</b>	<b>4</b>
1.1 Многопоточная обработка данных . . . . .	4
1.2 Алгоритм нахождения определителя матрицы методом миноров	4
1.3 Вывод . . . . .	5
<b>2 Конструкторская часть</b>	<b>6</b>
2.1 Разработка алгоритмов . . . . .	6
<b>3 Технологическая часть</b>	<b>8</b>
3.1 Средства реализации . . . . .	8
3.2 Реализация алгоритмов . . . . .	8
3.3 Функциональные тесты . . . . .	11
<b>4 Исследовательская часть</b>	<b>12</b>
4.1 Технические характеристики . . . . .	12
4.2 Демонстрация работы программы . . . . .	13
4.3 Время выполнения алгоритмов . . . . .	14
<b>Заключение</b>	<b>15</b>
<b>Список использованных источников</b>	<b>16</b>

# Введение

Сегодня компьютерам требуется выполнять все более трудоемкие вычисления. Одним из возможных решений, позволяющих увеличить производительность компьютера, является параллельное программирование. Параллельные вычисления на одном устройстве реализуются при помощи многопоточности — возможности процессора одновременно выполнять несколько потоков, поддерживаемых ОС.

Целью работы является исследование параллельного программирования на примере алгоритма нахождения определителя матрицы методом миноров. Для решения поставленной цели, необходимо выполнить следующие задачи:

- 1) Описать метод многопоточной обработки данных;
- 2) Разработать алгоритм нахождения определителя матрицы методом миноров;
- 3) Разработать параллельный алгоритм нахождения определителя матрицы методом миноров;
- 4) Сравнить разработанные алгоритмы.

# 1 Аналитическая часть

В этом разделе будет описан принцип многопоточной обработки данных и алгоритм многопоточной обработки данных.

## 1.1 Многопоточная обработка данных

Многопоточность - это способность процессора выполнять одновременно несколько потоков, используя ресурсы одного процессора. Поток представляет собой последовательность инструкций, которые могут выполняться параллельно с другими потоками в рамках одного процесса. Процесс - это программа, которая находится в состоянии выполнения. При запуске программы или приложения создается процесс. Процесс может состоять из одного или нескольких потоков, где каждый поток выполняет задачи, необходимые для работы приложения. Процесс завершается, когда все его потоки завершают свою работу.

В многопоточных программах необходимо учитывать, что если потоки запускаются последовательно и передают управление друг другу, то не получится полностью использовать потенциал многопоточности и получить выигрыш от параллельной обработки задач. Чтобы достичь максимальной эффективности, потоки должны выполняться параллельно, особенно для независимых по данным задач.

## 1.2 Алгоритм нахождения определителя матрицы методом миноров

Матрица — это набор чисел, записанный в виде прямоугольной таблицы. Строки и столбцы матрицы можно рассматривать как векторы. Матрица с одинаковым количеством строк и одинаковым количеством столбцов называется квадратной. Для обозначения элемента матрицы  $A$ , стоящего в  $i$ -той строке и в  $j$ -ом столбце используется запись  $A[i][j]$ .

Минором  $M_{ij}$  элемента  $a_{ij}$  матрицы  $A$   $n$ -го порядка называется определитель  $(n - 1)$ -го порядка, полученного из исходного определителя вычер-

киванием  $i$ -ой строки и  $j$ -го столбца [1]. Алгебраическим дополнением  $A_{ij}$  элемента  $a_{ij}$  матрицы  $A$   $n$ -го порядка называется число, равное произведению минора  $M_{ij}$  на  $(-1)^{i+j}$ :

$$A_{ij} = (-1)^{i+j} \cdot M_{ij}. \quad (1.1)$$

Таким образом определитель  $n$ -го порядка вычисляется с помощью метода понижения порядка — по формуле  $\det A = \sum_{j=1}^n a_{ij} A_{ij}$  ( $i$  фиксировано) — разложение по  $i$ -ой строке.

## 1.3 Вывод

В этом разделе был описан принцип многопоточной обработки данных и алгоритм многопоточной обработки данных.

## 2 Конструкторская часть

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

### 2.1 Разработка алгоритмов

На рис. 2.1-2.2 представлены схемы алгоритма линейной обработки данных, алгоритма многопоточной обработки данных.



Рис. 2.1 – Алгоритм линейной обработки данных

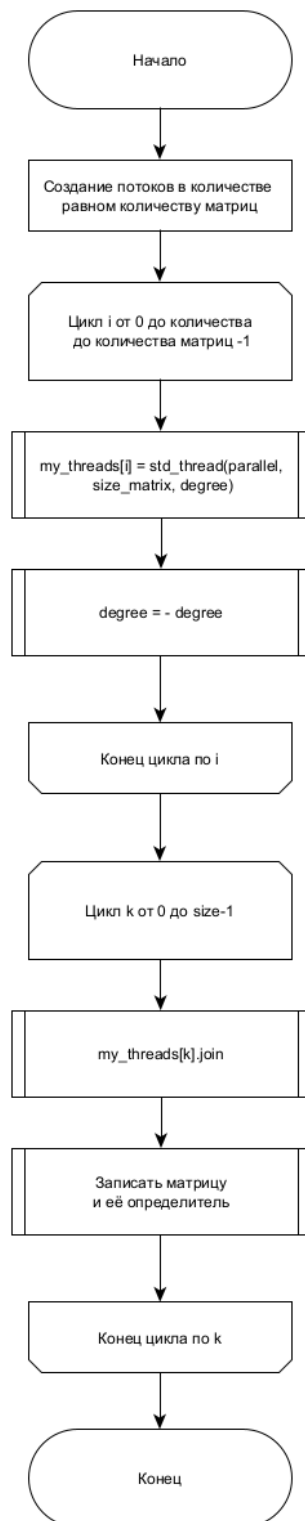


Рис. 2.2 – Алгоритм многопоточной обработки данных

## 3 Технологическая часть

В данном разделе будут рассмотрены средства реализации, а также представлены листинги реализаций алгоритма линейной обработки данных и алгоритма сногопоточной 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. Для создания потоков был использован класс *thread* [2].

### 3.2 Реализация алгоритмов

В листингах 3.1-3.2 представлены реализации алгоритмов линейной и многопоточной 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::thread threads[];
4     for (int i = 0; i < count; i++)
5         thread(i) = std::thread(stage_parallel, size_matrix, degree);
6     threads[i].push(size);
7     for (int i = 0; i < 3; i++)
8         threads[i].join();
9 }
```

### 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	Многопоточная
$\begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 3 \\ 2 & 4 & 0 \end{pmatrix}$	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
$\begin{pmatrix} 1 & 3 & 2 & 2 \\ 7 & 4 & 2 & 5 \\ 4 & 6 & 1 & 4 \\ 2 & 6 & 9 & 7 \end{pmatrix}$	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

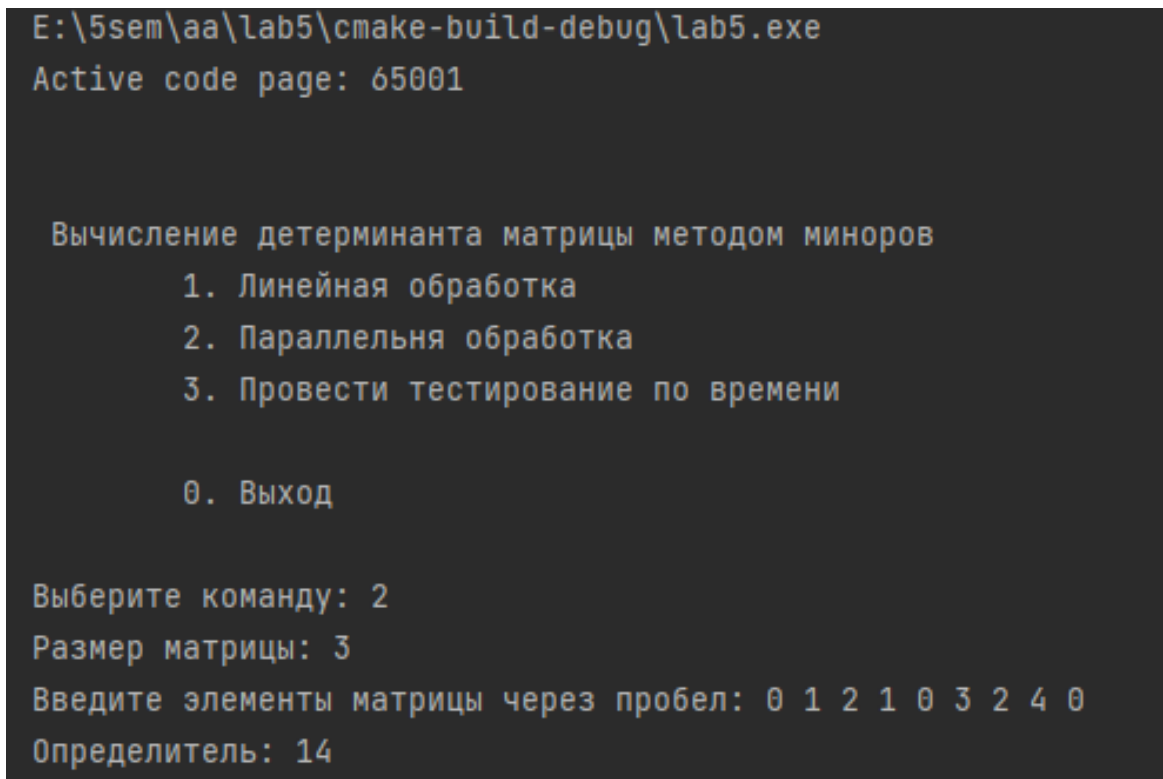
Технические характеристики устройства, на котором выполнялось тестирование представлены далее:

- 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.



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

Вычисление детерминанта матрицы методом миноров
    1. Линейная обработка
    2. Параллельная обработка
    3. Провести тестирование по времени

    0. Выход

Выберите команду: 2
Размер матрицы: 3
Введите элементы матрицы через пробел: 0 1 2 1 0 3 2 4 0
Определитель: 14
```

Рис. 4.1 – Демонстрация работы программы

## 4.3 Время выполнения алгоритмов

Результаты замеров приведены в таблице 4.1. Время указано в секундах.

Таблица 4.1 – Результаты замеров по времени

Размер матрицы	Линейная	Многопоточная
1	0.0369	0.0458
2	0.0937	0.0952
3	1.1308	1.0358
4	1.7972	1.0973
5	2.5906	1.4571
6	3.1332	1.8589
7	4.1956	2.0545
8	5.4542	2.2654
9	6.8004	2.7832
10	8.0206	3.1296

На рис. 4.2 представлен графический результат замеров времени работы линейного и многопоточного алгоритма поиска определителя.

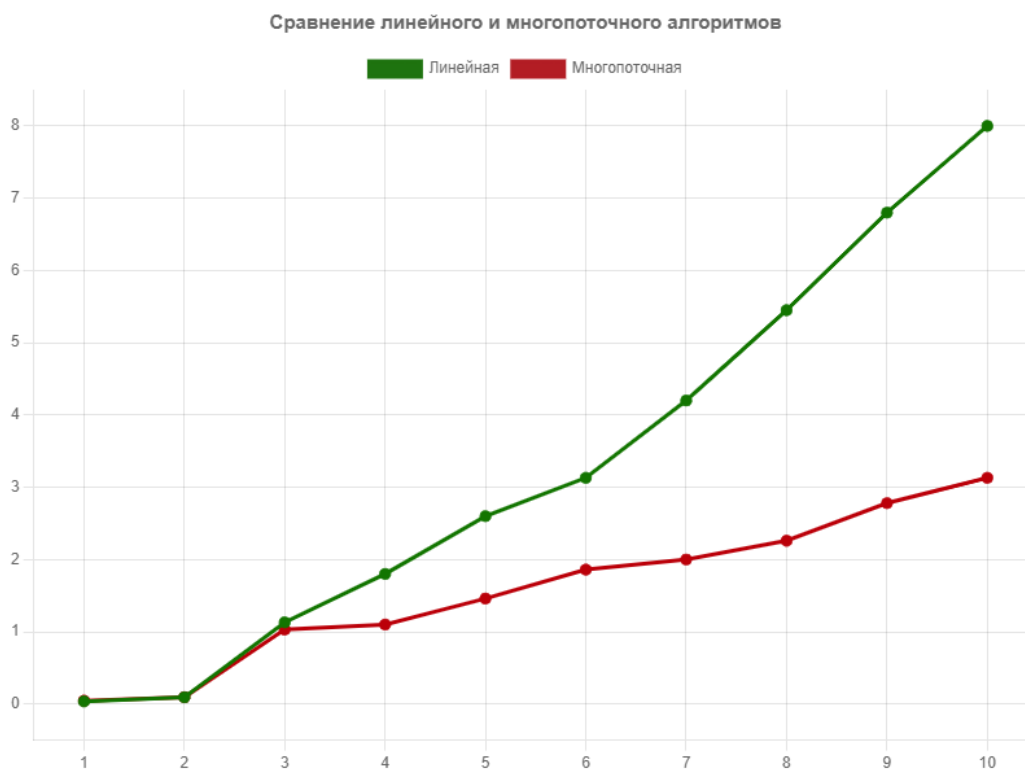


Рис. 4.2 – Сравнение времени работы алгоритмов

## Заключение

В результате было определено, что многопоточная обработка данных работает быстрее при размере матрицы больше двух засчет параллельного выполнения стадии поиска миноров для элементов матрицы.

Цель была достигнута, была исследована многопоточная обработка данных.

В ходе выполнения лабораторной работы были решены следующие задачи:

- 1) описан и реализован алгоритм многопоточной обработки данных;
- 2) реализован алгоритмы поиска определителя методом миноров;
- 3) проведено тестирование по времени для этих алгоритмов;
- 4) проведен сравнительный анализ по времени для этих алгоритмов.

## Список использованных источников

- [1] Minor - Encyclopedia of Mathematics [Электронный ресурс]. Режим доступа: <https://encyclopediaofmath.org/index.php?title=Minor&oldid=30176> (дата обращения: 05.01.2024).
  
- [2] thread Class [Электронный ресурс]. Режим доступа: <https://learn.microsoft.com/en-us/cpp/standard-library/thread-class?view=msvc-170> (дата обращения: 05.01.2024).