



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 (ИУ7)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.04.03 ПРОГРАММНАЯ ИНЖЕНЕРИЯ

**О Т Ч Е Т**

По лабораторной работе № 4

Название: Параллельные вычисления

Дисциплина: Анализ Алгоритмов

Студент

ИУ7-52Б

\_\_\_\_\_  
(Группа)

Н.А. Гарасев

\_\_\_\_\_  
(Подпись, дата)

\_\_\_\_\_  
(И.О. Фамилия)

Преподаватель

Л.Л. Волкова

\_\_\_\_\_  
(Подпись, дата)

\_\_\_\_\_  
(И.О. Фамилия)

Москва, 2020

# Оглавление

О Т Ч Е Т .....	1
.....	1
Введение.....	3
1. Аналитическая часть.....	4
1.2. Многопоточность .....	4
2. Конструкторская часть .....	7
2.1. Схемы алгоритмов .....	7
2.2. Распараллеливание алгоритма .....	9
3. Технологическая часть .....	10
3.1. Реализация алгоритмов.....	10
3.2. Тестирование функций .....	13
4. Экспериментальная часть.....	14
4.1. Сравнение алгоритмов по времени работы реализаций .....	14
Заключение .....	18
Список литературы .....	19

## **Введение**

**Цель** лабораторной работы: изучить и применить на практике возможности параллельных вычислений на примере умножения матриц.

Алгоритмы перемножения матриц активно применяются во всех областях, где используется линейная алгебра.

- 1) В компьютерной графике.
- 2) В физике.
- 3) В экономике.

В ходе выполнения лабораторной работы требуется решить следующие **задачи**.

- 1) Реализовать алгоритм умножения матриц.
- 2) Реализовать две параллельные версии алгоритма.
- 3) Провести сравнительный анализ алгоритмов умножения матриц.
- 4) Сравнить алгоритмы по затраченным ресурсам.
- 5) Проанализировать полученные результаты и сделать вывод о эффективности реализованных алгоритмов.

## 1. Аналитическая часть

Умножение матриц — одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется произведением матриц. Матрицы **A** и **B** могут быть перемножены, если они совместимы в том смысле, что число столбцов матрицы **A** равно числу строк **B**.

### 1.1. Классическое умножение матриц

Ниже на рис. 1 представлена формула умножения. Пусть даны две прямоугольные матрица **A** и **B** размерами  $n * m$  и  $m * k$  соответственно. Тогда матрица **C** имеет размеры  $n * k$  и является произведением матриц **A** и **B**, каждый элемент  $c_{ij}$  которой считается по формуле, представленной на рис. 1.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \cdots & a_{lm} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix}.$$
$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \cdots & c_{ln} \end{bmatrix},$$

Рисунок 1. Перемножение матриц **A** и **B** дает в результате матрицу **C**

### 1.2. Многопоточность

**Поток выполнения** — наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Реализация потоков выполнения и процессов в разных операционных системах отличается друг от друга, но в большинстве случаев поток выполнения находится внутри

процесса. Несколько потоков выполнения могут существовать в рамках одного и того же процесса и совместно использовать ресурсы, такие как память, тогда как процессы не разделяют этих ресурсов. В частности, потоки выполнения разделяют инструкции процесса (его код) и его контекст (значения переменных, которые они имеют в любой момент времени).

На одном процессоре многопоточность обычно происходит путём временного мультиплексирования (как и в случае многозадачности): процессор переключается между разными потоками выполнения. Это переключение контекста обычно происходит достаточно часто, чтобы пользователь воспринимал выполнение потоков или задач как одновременное. В многопроцессорных и многоядерных системах потоки или задачи могут реально выполняться одновременно, при этом каждый процессор или ядро обрабатывает отдельный поток или задачу.

Потоки возникли в операционных системах как средство распараллеливания вычислений.

Параллельное выполнение нескольких работ в рамках одного интерактивного приложения повышает эффективность работы пользователя. Так, при работе с текстовым редактором желательно иметь возможность совмещать набор нового текста с такими продолжительными по времени операциями, как переформатирование значительной части текста, печать документа или его сохранение на локальном или удаленном диске. Еще одним примером необходимости распараллеливания является сетевой сервер баз данных. В этом случае параллелизм желателен как для обслуживания различных запросов к базе данных, так и для более быстрого выполнения отдельного запроса за счет одновременного просмотра различных записей базы. Именно для этих целей современные ОС предлагают механизм многопоточной обработки (multithreading). Понятию «поток» соответствует последовательный переход процессора от одной команды программы к другой. ОС распределяет процессорное время между потоками. Процессу ОС

назначает адресное пространство и набор ресурсов, которые совместно используются всеми его потоками.

Создание потоков требует от ОС меньших накладных расходов, чем процессов. В отличие от процессов, которые принадлежат разным, вообще говоря, конкурирующим приложениям, все потоки одного процесса всегда принадлежат одному приложению, поэтому ОС изолирует потоки в гораздо меньшей степени, нежели процессы в традиционной мультипрограммной системе. Все потоки одного процесса используют общие файлы, таймеры, устройства, одну и ту же область оперативной памяти, одно и то же адресное пространство. Это означает, что они разделяют одни и те же глобальные переменные. Поскольку каждый поток может иметь доступ к любому виртуальному адресу процесса, один поток может использовать стек другого потока. Между потоками одного процесса нет полной защиты, потому что, во-первых, это невозможно, а во-вторых, не нужно. Чтобы организовать взаимодействие и обмен данными, потокам вовсе не требуется обращаться к ОС, им достаточно использовать общую память — один поток записывает данные, а другой читает их. С другой стороны, потоки разных процессов по-прежнему хорошо защищены друг от друга.

Широко используемый подход состоит и в применении тех или иных библиотек, обеспечивающих определенный программный интерфейс (application programming interface, API) для разработки параллельных программ. В рамках такого подхода наиболее известны Windows Thread API. Однако первый способ применим только для ОС семейства Microsoft Windows, а второй вариант API является достаточно трудоемким для использования и имеет низкоуровневый характер [2].

## 2. Конструкторская часть

На вход алгоритмы принимают две матрицы. На выходе выдают матрицу — результат произведения двух этих матриц.

### 2.1. Схемы алгоритмов

На рис. 2-4 приведены схемы алгоритмов:

- 1) стандартный алгоритм перемножения матриц;
- 2) первая версия распараллеливания алгоритма перемножения матриц;
- 3) вторая версия распараллеливания алгоритма перемножения матриц.

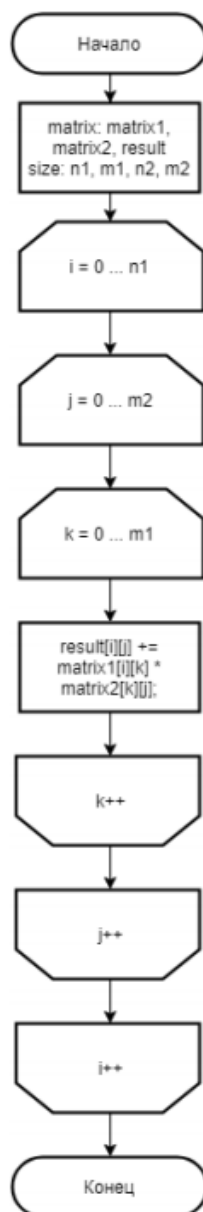


Рисунок 2. Стандартный алгоритм перемножения матриц

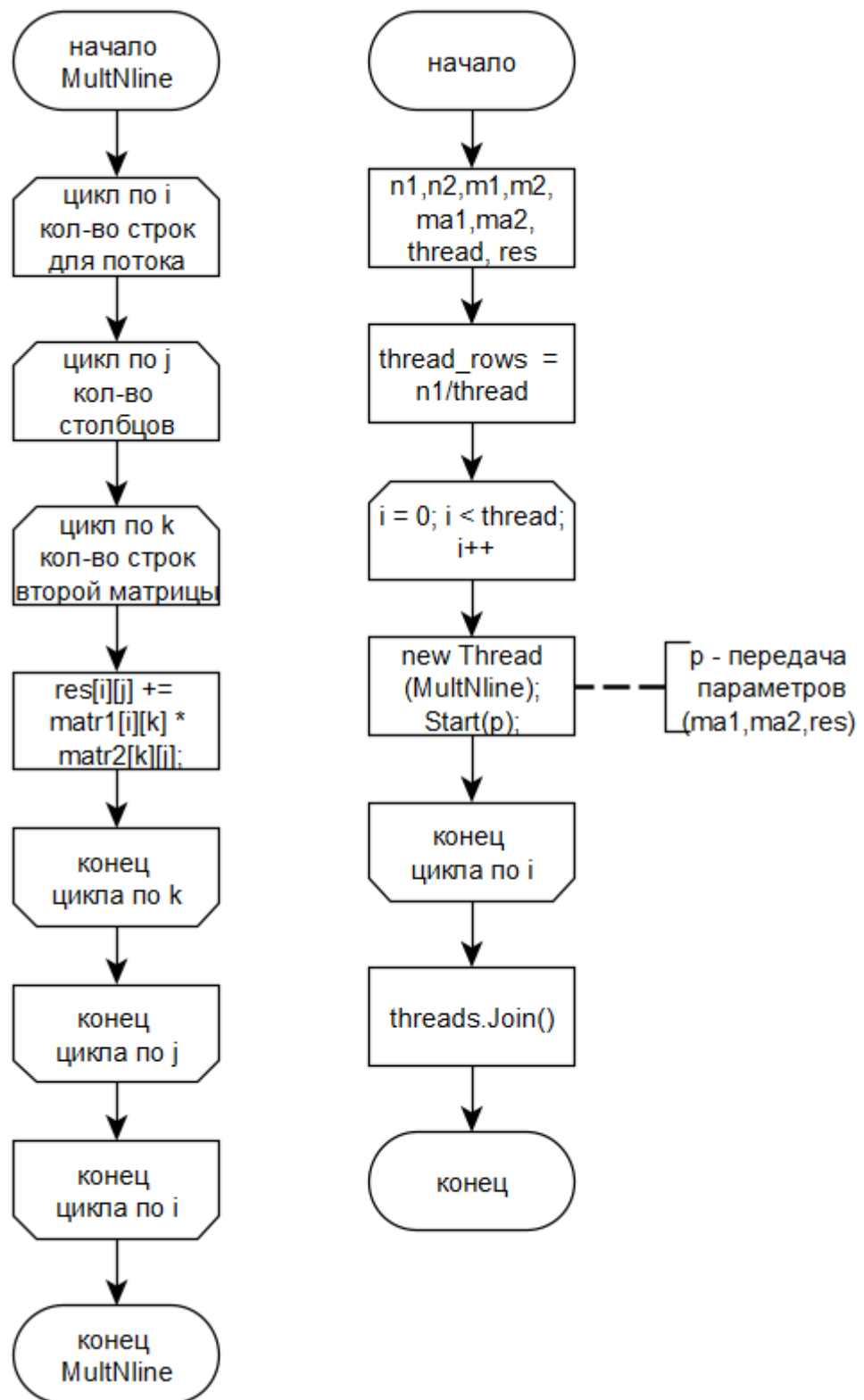


Рисунок 3. Первая версия распараллеливания алгоритма перемножения матриц



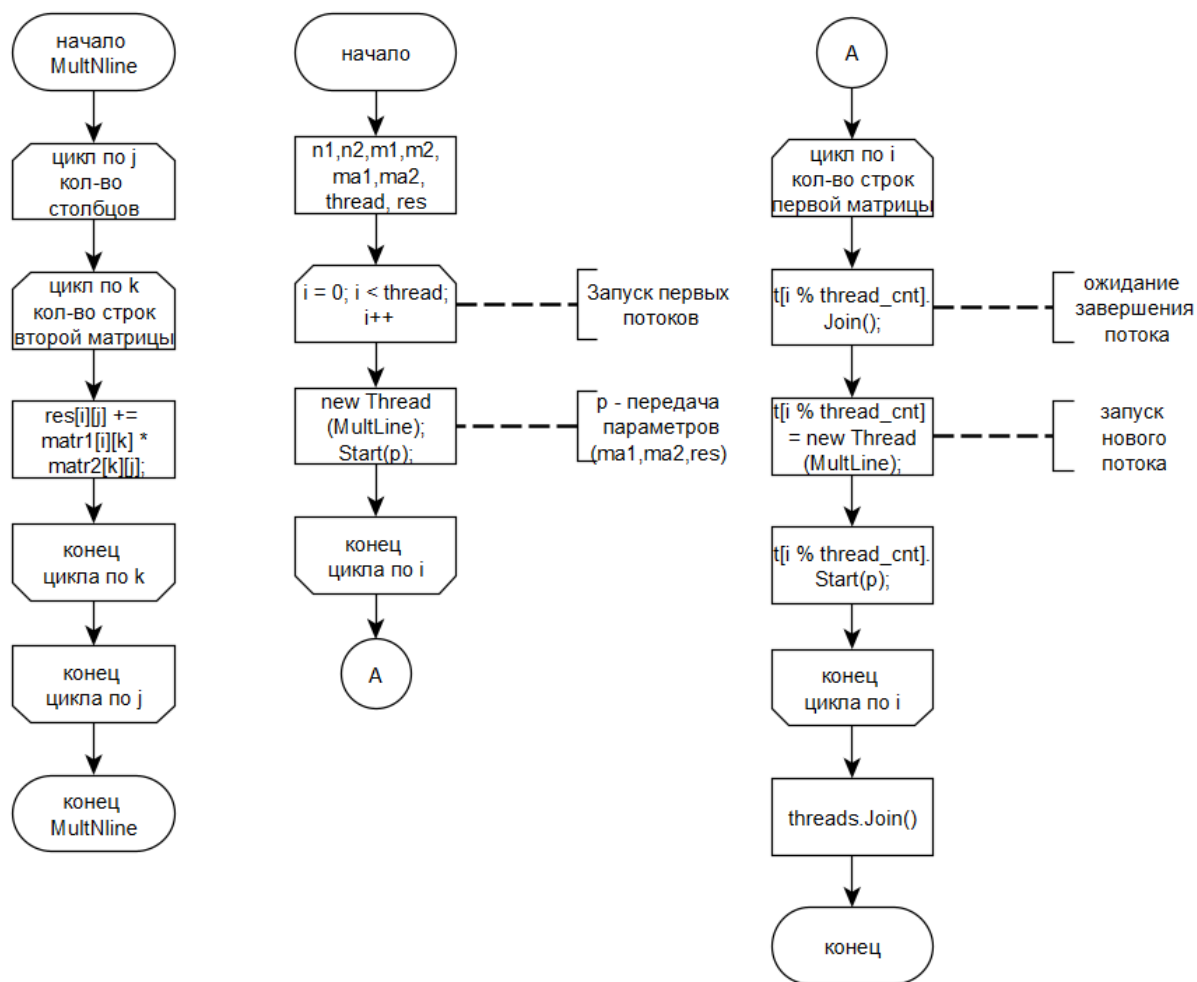


Рисунок 4. Вторая версия распараллеливания алгоритма перемножения матриц

## 2.2. Распараллеливание алгоритма

В программе представлены два подхода к распараллеливанию алгоритма умножения матриц. Суть первого подхода заключается в том, что общее количество строк первой матрицы разделяется по потокам на равное количество. Подход второго распараллеливания заключается в том, что нам даются строки матрицы, а мы их по очереди посылаем в разные потоки.

### 3. Технологическая часть

В качестве языка программирования был выбран C#, т.к. на сегодняшний момент язык программирования C# один из самых мощных, быстро развивающихся и востребованных языков в ИТ-отрасли. В настоящий момент на нем пишутся самые различные приложения: от небольших десктопных программ до крупных веб-порталов и веб-сервисов, обслуживающих ежедневно миллионы пользователей. Основной функционал для использования потоков в приложении сосредоточен в пространстве имен **System.Threading**. Так же для лабораторной работы использовалось **System.Diagnostics**.

#### 3.1 Реализация алгоритмов

В листингах 1-3 представлена реализация алгоритмов умножения матриц. В листинге 4 представлена функция для замера времени выполнения заданной функции.

Листинг 1. Реализация алгоритма умножения матриц

```
public int[][] Mult(int[][] ma1, int[][] ma2, int thread_count)
{
    int n1 = ma1.Length;
    int n2 = ma2.Length;

    if (n1 == 0 || n2 == 0)
        return null;

    int m1 = ma1[0].Length;
    int m2 = ma2[0].Length;

    if (m1 != n2)
        return null;

    int[][] res = new int[n1][];
    for (int i = 0; i < n1; i++)
        res[i] = new int[m2];

    for (int i = 0; i < n1; i++)
        for (int j = 0; j < m2; j++)
            for (int k = 0; k < m1; k++)
                res[i][j] += ma1[i][k] * ma2[k][j];

    return res;
}
```

## Листинг 2. Реализация первого распараллеливания умножения матриц

```
public void MultNline(object obj)
{
    Params p = (Params)obj;

    for (int i = p.str; i < p.end; i++)
        for (int j = 0; j < p.matr2[0].Length; j++)
            for (int k = 0; k < p.matr1[0].Length; k++)
                p.res[i][j] += p.matr1[i][k] * p.matr2[k][j];
}

public int[][] MultParal1(int[][] ma1, int[][] ma2, int thread_cnt)
{
    int n1 = ma1.Length;
    int n2 = ma2.Length;

    if (n1 == 0 || n2 == 0)
        return null;

    int m1 = ma1[0].Length;
    int m2 = ma2[0].Length;

    if (m1 != n2)
        return null;

    int[][] res = new int[n1][];
    for (int i = 0; i < n1; i++)
        res[i] = new int[m2];

    Thread[] t = new Thread[thread_cnt];

    int thread_rows = n1 / thread_cnt;
    int str = 0;

    for (int i = 0; i < thread_cnt; i++)
    {
        int end = str + thread_rows;
        if (i == thread_cnt - 1)
            end = n1;

        Params p = new Params(res, ma1, ma2, str, end);

        t[i] = new Thread(MultNline);
        t[i].Start(p);

        str = end;
    }

    foreach (Thread thread in t)
    {
        thread.Join();
    }
    return res;
}
```

## Листинг 3. Реализация второго распараллеливания умножения матриц

```
public void MultLine(object obj)
{
    Params p = (Params)obj;

    for (int j = 0; j < p.matr2[0].Length; j++)
        for (int k = 0; k < p.matr1[0].Length; k++)
```

```

        p.res[p.str][j] += p.matr1[p.str][k] * p.matr2[k][j];
    }

    public int[][] MultPara12(int[][] ma1, int[][] ma2, int thread_cnt)
    {
        int n1 = ma1.Length;
        int n2 = ma2.Length;

        if (n1 == 0 || n2 == 0)
            return null;

        int m1 = ma1[0].Length;
        int m2 = ma2[0].Length;

        if (m1 != n2)
            return null;

        int[][] res = new int[n1][];
        for (int i = 0; i < n1; i++)
            res[i] = new int[m2];

        Thread[] t = new Thread[thread_cnt];

        for (int i = 0; i < thread_cnt; i++)
        {
            Params p = new Params(res, ma1, ma2, i);
            t[i] = new Thread(MultLine);
            t[i].Start(p);
        }

        for (int i = thread_cnt; i < n1; i++)
        {
            Params p = new Params(res, ma1, ma2, i);

            t[i % thread_cnt].Join();
            t[i % thread_cnt] = new Thread(MultLine);
            t[i % thread_cnt].Start(p);
        }

        foreach (Thread thread in t)
        {
            thread.Join();
        }
        return res;
    }
}

```

Листинг 4. Функция подсчета среднего времени выполнения алгоритма умножения матриц.

```

public static void Time(Func<int[][], int[][], int, int[][],> MultFunc, int thread, int n)
{
    Stopwatch stopWatch = new Stopwatch();
    TimeSpan ts;
    for (int size = 100; size <= 800; size += 100)
    {
        ts = new TimeSpan();
        for (int repeat = 1; repeat <= n; repeat++)
        {
            int[][] a = FillMatr(size, size);
            int[][] b = FillMatr(size, size);

            stopWatch.Start();

```

```

        MultFunc(a, b, thread);

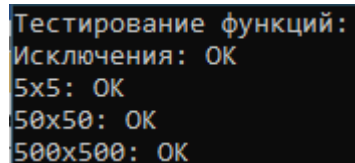
        stopWatch.Stop();
        ts += stopWatch.Elapsed;

    }
    ts = ts / n;
    string elapsedTime = String.Format("{0:00}:{1:00}.{2:00}",
        ts.Minutes, ts.Seconds,
        ts.Milliseconds / 10);
    Console.WriteLine(" " + size.ToString() + ": " + elapsedTime);
}
}

```

## 3.2. Тестирование функций

Для модульного тестирования реализованных алгоритмах (см. листинги 1-3) все функции протестированы на случайных входных данных.



```

Тестирование функций:
Исключения: OK
5x5: OK
50x50: OK
500x500: OK

```

Рисунок 5. Тестирование функций

Все тесты были пройдены успешно для всех реализованных алгоритмов сортировки.

# 1. Экспериментальная часть

Сравним реализованные алгоритмы по времени.

## 4.1. Сравнение алгоритмов по времени работы реализаций

Для сравнения в программе необходимо провести замеры процессорного времени для выполнения  $n$  – количества вычислений для матриц одинакового размера. Данное время замеров представлено на рисунке 6.

```
Обычное умножение:  
100: 00:00.00  
200: 00:00.04  
300: 00:00.22  
400: 00:00.71  
500: 00:01.84  
600: 00:03.32  
700: 00:05.98  
800: 00:09.99
```

Рисунок 6. Среднее время простого умножения матриц

Далее на рисунках 7-10 представлены результаты для распараллеленных алгоритмов. Результаты разделены по количеству потоков.

```
Умножение разделенные на группы строки по потокам:  
1 поток:  
100: 00:00.01  
200: 00:00.07  
300: 00:00.26  
400: 00:00.73  
500: 00:01.73  
600: 00:03.45  
700: 00:06.74  
800: 00:11.53  
2 поток:  
100: 00:00.05  
200: 00:00.11  
300: 00:00.23  
400: 00:00.48  
500: 00:00.97  
600: 00:01.84  
700: 00:03.23  
800: 00:05.48
```

Рисунок 7. Среднее время первого подхода распараллеливания для 1ого  
и 2х потоков

```
4 поток:
100: 00:00.06
200: 00:00.14
300: 00:00.21
400: 00:00.41
500: 00:00.75
600: 00:01.26
700: 00:02.04
800: 00:03.30
8 поток:
100: 00:00.07
200: 00:00.20
300: 00:00.35
400: 00:00.54
500: 00:00.88
600: 00:01.46
700: 00:02.38
800: 00:03.92
16 поток:
100: 00:00.23
200: 00:00.40
300: 00:00.58
400: 00:00.83
500: 00:01.21
600: 00:01.88
700: 00:02.85
800: 00:04.26
```

Рисунок 8. Среднее время первого подхода распараллеливания для 4х,  
8ми и 16ти потоков

```
Умножение одна строка - один поток и так по кругу по потокам:
1 поток:
100: 00:00.47
200: 00:01.52
300: 00:03.33
400: 00:05.96
500: 00:09.67
600: 00:14.84
700: 00:21.84
800: 00:31.17
2 поток:
100: 00:00.72
200: 00:02.76
300: 00:05.90
400: 00:10.05
500: 00:14.76
600: 00:20.59
700: 00:27.48
800: 00:36.40
```

Рисунок 9. Среднее время второго подхода распараллеливания для 1ого  
и 2х потоков

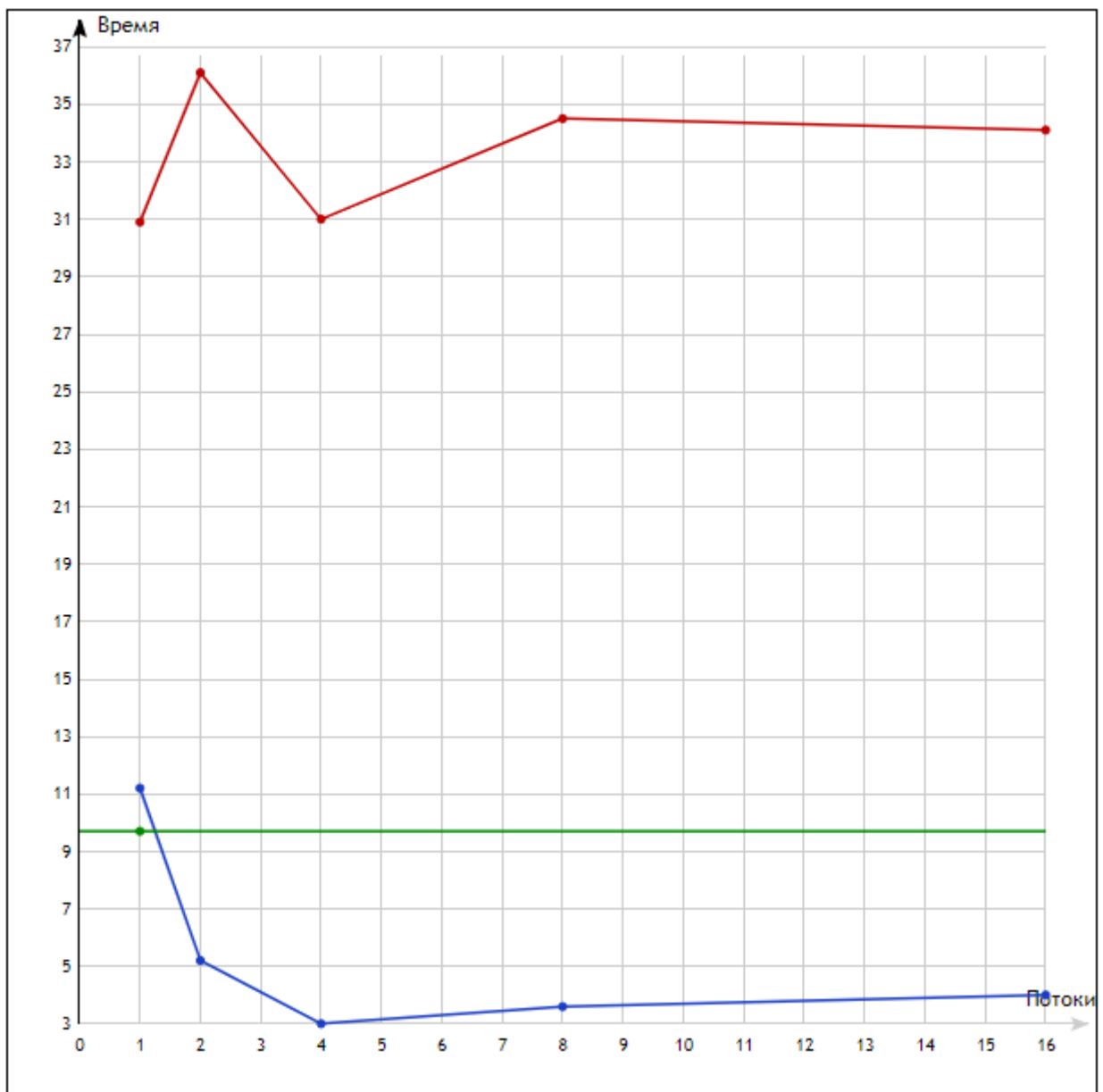
```
4 поток:
100: 00:00.83
200: 00:02.49
300: 00:05.03
400: 00:08.59
500: 00:12.94
600: 00:17.63
700: 00:24.10
800: 00:31.27
8 поток:
100: 00:00.91
200: 00:02.85
300: 00:05.77
400: 00:09.59
500: 00:14.59
600: 00:20.05
700: 00:27.05
800: 00:34.81
16 поток:
100: 00:00.97
200: 00:03.63
300: 00:06.38
400: 00:10.03
500: 00:14.40
600: 00:20.12
700: 00:26.62
800: 00:34.41
Hello World!
```

Рисунок 10. Среднее время второго подхода распараллеливания для 4х, 8ми и 16ти потоков

Количество логических процессоров экспериментальной машины — равно четырем.

На рисунке 11 представлены графики зависимости времени от количества потоков на примере реализаций алгоритмов умножения матриц. Обычное умножение без распараллеливания потоков изображено зеленой линией. Первая реализация по группам строк показана синей линией. Вторая реализация умножения матриц с распараллеливанием изображена красной линией.





**Вывод:** наглядно видно, что первый подход распараллеливания (деление на группы строк) выполняет свою задачу и уменьшает время выполнения алгоритма. Скорость выполнения напрямую зависит от количества потоков, но после 4х потоков рост скорости останавливается. Это связано с тем, что количество логических процессоров экспериментальной машины — равно четырем. Во втором случае (умножение строки в потоке) все намного хуже. Время выполнения алгоритма хуже, обычного умножения. Это говорит нам о том, что при данных условиях производить параллельные вычисления построчно более затратно по времени, чем выделять потоки для этого умножения.

## **Заключение**

В ходе работы были изучены и реализованы распараллеленные алгоритмы умножения матриц.

Был сделан вывод для использования распараллеливания для алгоритмов, основанный на временных замерах реализованных алгоритмов

Цель работы достигнута. Получены практические навыки реализации алгоритмов распараллеливания, а также проведена исследовательская работа по вычислению и сравнению этих алгоритмов.

## Список литературы

1. Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход. –М.: Техносфера, 2017. – 267 с.
2. Сайт о программировании, С# [Электронный ресурс]. Режим доступа: <https://metanit.com/>, свободный (дата обращения: 07.12.20).