**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»

# Лабораторная работа № 4

**Дисциплина** Анализ алгоритмов

**Тема** Параллельное умножение матриц

**Студент** Зуев Никита

**Группа** ИУ7-51Б

**Оценка (баллы)** _____

**Преподаватель** Волкова Л.Л, Строганов Ю.В

Москва — 2020 г.

# Оглавление

# Введение

Цель работы: изучение возможности параллельных вычислений и использование такого подхода на практике. Реализация парралельного алгоритма Винограда умножения матриц.

**Задачами** данной лабораторной работы являются:

1. изучение параллельных вычислений;
2. реализация алгоритма винограда с возможностью распараллеливания вычислений;
3. экспериментальное подтверждение различий во временно́й эффективности алгоритмов, задействующих разное количество потоков;
4. описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

# 1. Аналитическая часть

## 1.1 Алгоритм Винограда

Матрицей A размера $[m * n]$ называется прямоугольная таблица чисел, функций или алгебраических выражений, содержащая m строк и n столбцов. Числа m и n определяют размер матрицы.[2] Если $m_2 = n_1$, то эти две матрицы можно перемножить. У произведения будет столько же строк, сколько в первой матрице, и столько же столбцов, сколько во второй. [2]

Пусть даны две прямоугольные матрицы A и B размеров $[m * n]$ и $[n * k]$ соответственно. В результате произведение матриц A и B получим матрицу C размера $[m * k]$.

$c_{i,j} = \sum_{r=1}^{n} a_{i,r} \cdot b_{r,j}$ называется произведением матриц A и B [2].

## 1.2 Алгоритм Винограда

Если рассмотреть результат умножения двух матриц, то видно, что каждый элемент в нём представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Можно заметить также, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее. [1].

Рассмотрим два вектора $V = (v1, v2, v3, v4)$ и $W = (w1, w2, w3, w4)$.

Их скалярное произведение равно (1.1)

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4 \tag{1.1}$$

Равенство (1.1) можно переписать в виде (1.2)

$$V \cdot W = (v_1 + w_2) \cdot (v_2 + w_1) + (v_3 + w_4) \cdot (v_4 + w_3) - v_1 \cdot v_2 - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4 \tag{1.2}$$

Выражение в правой части последнего равенства допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй. Это означает, что над предварительно обработанными элементами нам придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

### 1.2.1 Худший случай

В случае, если m1(количество столбцов в первой матрице ) нечетное , необходимо дополнительно к каждому элементу результирующей матрице прибавить произведение

$$M1[i = 0..n1 - 1][m1 - 1] * M2[m1 - 1][j = 0..m2] \tag{1.3}$$

### 1.2.2 Параллельный алгоритм Винограда

Самая трудоемкая часть алгоритма - тройной цикл, и возникающая в следствии сложность $O(MNK)$. Для максимального прироста эффективности следует распараллелить именно эту часть

Вычисление элементов очередной строки не зависит от реузьтата умножения других строк. Потоки смогут работать независимо, не ожидая разблокировки доступа к данным.

## 1.3 Параллельное программирование

Существуют различные способы реализации параллельных вычислений. Например, каждый вычислительный процесс может быть реализован в виде процесса операционной системы, либо же вычислительные процессы могут представлять собой набор потоков выполнения внутри одного процесса ОС. Параллельные программы могут физически исполняться либо последовательно на единственном процессоре — перемежая по очереди шаги выполнения каждого вычислительного процесса, либо параллельно — выделяя каждому вычислительному процессу один или несколько

процессоров (находящихся рядом или распределённых в компьютерную сеть).

Основная сложность при проектировании параллельных программ — обеспечить правильную последовательность взаимодействий между различными вычислительными процессами, а также координацию ресурсов, разделяемых между процессами.[3]

### 1.3.1 Организация взаимодействия параллельных потоков

Потоки исполняются в общем адресном пространстве параллельной программы. Как результат, взаимодействие параллельных потоков можно организовать через использование общих данных, являющихся доступными для всех потоков. Наиболее простая ситуация состоит в использовании общих данных только для чтения. В случае же, когда общие данные могут изменяться несколькими потоками, необходимы специальные усилия для организации правильного взаимодействия.

## 1.4 Вывод

Был рассмотрен алгоритм Винограда и возможность его оптимизации с помощью распараллеливания потоков. Была рассмотрена технология параллельного программирования и организация взаимодействия параллельных потоков.

# 2. Конструкторская часть

**Требования к вводу:** На вход подаются две матрицы
**Требования к программе:**

- корректное умножение двух матриц;
- при матрицах неправилыных размеров программа не должна аварийно завершаться.

## 2.1 Схемы алгоритмов

В данной части будет рассмотрена схема оптимизировнного алгоритма Винограда 2.1 и выбранный спрособ распараллеливания.

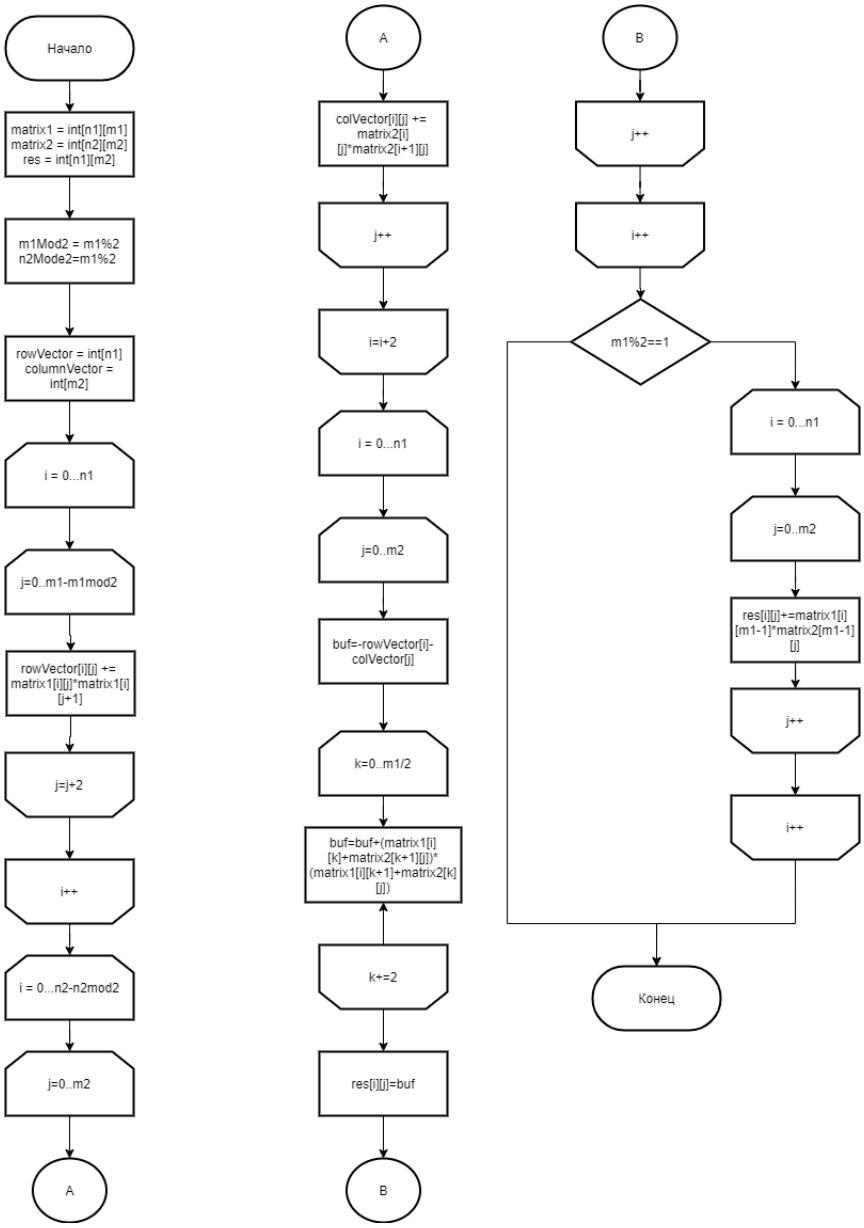


Рис. 2.1: Схема оптимизированного алгоритма умножения матриц Винограда

## 2.2 Распараллеливание программы

Распраллелим участок с самым большим количеством вычислений. В данной реализации алгоритма таким участком будет являться тройной цикл, вычисляющий значения элементов результирающей матрицы(На 2.1 находиться между участков A и B).

## 2.3 Вывод

В данном разделе была рассмотрена схема алгоритма Винограда и способ ее распараллеливания.

# 3. Технологическая часть

## 3.1 Выбор ЯП

В качестве языка программирования был выбрант, так как, начиная с версии 1.2 Java поддерживает нативные потоки, и с тех пор они используются по умолчанию. Многопоточность реализована с помощью интефейся Runnable.

## 3.2 Сведения о модулях программы

Программа состоит из:

- TimeMeasureLab4.java - главный файл программы, в котором располагается точка входа в программу и функция замера времени;
- Matrix.java - файл класса, в которм находится обычный алгоритм Винограда;
- MatrixParallel.java - файл класса в котором находится распаралелленый алгоритм Винограда.
- TestLav4.java - файл с модульными тестами

## 3.3 Листинг кода алгоритмов

Ниже представлена реализация алгоритмов:

- Реализация алгоритма Виногарада 3.1
- Реалзиация алгоритма Винограда с использованием потоков 3.2

- Замер времени работы алгоритмов 3.3

Листинг 3.1: Алгоритм Винограда

```
1  package lab4;
2
3  public class Matrix {
4
5    public static int[][] MultVinOpt(int[][] matr1, int[][]
       matr2)
6    {
7      int n1 = matr1.length;
8      int n2 = matr2.length;
9
10     if (n1 == 0 || n2 == 0)
11     return null;
12
13     int m1 = matr1[0].length;
14     int m2 = matr2[0].length;
15
16     if (m1 != n2 || m1 == 0 || m2 == 0)
17     return null;
18
19     int[] rowVector = new int[n1];
20     int[] colVector = new int[m2];
21
22     int[][] res = new int[n1][];
23     for (int i = 0; i < n1; i++)
24     res[i] = new int[m2];
25
26     int m1Mod2 = m1 % 2;
27     int n2Mod2 = n2 % 2;
28
29     for (int i = 0; i < n1; i++)
30     {
31       for (int j = 0; j < (m1 - m1Mod2); j += 2)
32       {
33         rowVector[i] += matr1[i][j] * matr1[i][j + 1];
34       }
35     }
36
```

```
37      for (int i = 0; i < m2; i++)
38      {
39        for (int j = 0; j < (n2 - n2Mod2); j += 2)
40        {
41          colVector[i] += matr2[j][i] * matr2[j + 1][i];
42        }
43      }
44
45      for (int i = 0; i < n1; i++)
46      {
47        for (int j = 0; j < m2; j++)
48        {
49          int buff = -(rowVector[i] + colVector[j]);
50          for (int k = 0; k < (m1 - m1Mod2); k += 2)
51          {
52            buff += (matr1[i][k + 1] + matr2[k][j]) * (matr1[
                  i][k] + matr2[k + 1][j]);
53          }
54          res[i][j] = buff;
55        }
56      }
57
58      if (m1Mod2 == 1)
59      {
60        int m1Min_1 = m1 - 1;
61        for (int i = 0; i < n1; i++)
62        {
63          for (int j = 0; j < m2; j++)
64          {
65            res[i][j] += matr1[i][m1Min_1] * matr2[m1Min_1][j
                  ];
66          }
67        }
68      }
69
70      return res;
71    }
72  }
```

Листинг 3.2: Параллельный алгоритм Винограда

```
1  package lab4;
2
3  import java.util.Arrays;
4
5  public class MatrixParallel {
6    public static class ParallelCounting implements Runnable
       {
7      private int[][] res, matr1, matr2;
8      private int[] rowVector, colVector;
9      private int n1, m1, m2, start;
10
11     public  ParallelCounting(MatrixInfoDto dto) {
12       res = dto.getRes();
13       matr1 = dto.getLeft();
14       matr2 = dto.getRight();
15       rowVector = dto.getRowvector();
16       colVector = dto.getColvector();
17       n1 = dto.getN1();
18       start = dto.getN2();
19       m1 = dto.getM1();
20       m2 = dto.getM2();
21     }
22
23     @Override
24     public void run() {
25       for (int i = start; i < n1; i++)
26       {
27         for (int j = 0; j < m2; j++)
28         {
29           int buff = -(rowVector[i] + colVector[j]);
30           for (int k = 0; k < (m1 - m1%2); k += 2)
31           {
32             buff += (matr1[i][k + 1] + matr2[k][j]) * (
                   matr1[i][k] + matr2[k + 1][j]);
33           }
34           res[i][j] = buff;
35         }
36       }
37     }
```

```
38  }
39
40  public static int[][] MultVinOpt(int[][] matr1, int[][]
        matr2, int thread_number) throws InterruptedException
        {
41    int n1 = matr1.length;
42    int n2 = matr2.length;
43
44    if (n1 == 0 || n2 == 0)
45    return null;
46
47    int m1 = matr1[0].length;
48    int m2 = matr2[0].length;
49
50    if (m1 != n2 || m1 == 0 || m2 == 0)
51    return null;
52
53    int[] rowVector = new int[n1];
54    int[] colVector = new int[m2];
55
56    int[][] res = new int[n1][];
57    for (int i = 0; i < n1; i++)
58    res[i] = new int[m2];
59
60    int m1Mod2 = m1 % 2;
61    int n2Mod2 = n2 % 2;
62
63    for (int i = 0; i < n1; i++)
64    {
65      for (int j = 0; j < (m1 - m1Mod2); j += 2)
66      {
67        rowVector[i] += matr1[i][j] * matr1[i][j + 1];
68      }
69    }
70
71    for (int i = 0; i < m2; i++)
72    {
73      for (int j = 0; j < (n2 - n2Mod2); j += 2)
74      {
75        colVector[i] += matr2[j][i] * matr2[j + 1][i];
```

```
76          }
77      }
78
79      Thread[] threads = new Thread[thread_number];
80      int row_start = 0;
81      int rowsPerThred = threads.length/thread_number;
82      for (int i  = 0; i < thread_number; i++) {
83        int row_end = row_start + rowsPerThred;
84        if (i == thread_number - 1)
85        row_end = n1;
86
87        MatrixInfoDto dto = new MatrixInfoDto();
88        dto.setRes(res);
89        dto.setLeft(matr1);
90        dto.setRight(matr2);
91        dto.setRowvector(rowVector);
92        dto.setColvector(colVector);
93        dto.setM1(m1);
94        dto.setM2(m2);
95        dto.setN2(row_start);
96        dto.setN1(row_end);
97
98        MatrixParallel.ParallelCounting e = new
              MatrixParallel.ParallelCounting(dto);
99        threads[i] = new Thread(e);
100
101       threads[i].start();
102
103       row_start = row_end;
104     }
105
106     for (int i = 0; i < threads.length; i++) {
107       threads[i].join();
108     }
109
110
111     if (m1Mod2 == 1)
112     {
113       int m1Min_1 = m1 - 1;
114       for (int i = 0; i < n1; i++)
```

```
115         {
116           for (int j = 0; j < m2; j++)
117           {
118             res[i][j] += matr1[i][m1Min_1] * matr2[m1Min_1][j
                    ];
119           }
120         }
121       }
122
123       return res;
124     }
125 }
```

Листинг 3.3: Замер времени работы алгоритмов

```
1   package lab4;
2
3   import org.openjdk.jmh.annotations.*;
4   import org.openjdk.jmh.results.format.ResultFormatType;
5   import org.openjdk.jmh.runner.Runner;
6   import org.openjdk.jmh.runner.RunnerException;
7   import org.openjdk.jmh.runner.options.Options;
8   import org.openjdk.jmh.runner.options.OptionsBuilder;
9
10  import java.io.IOException;
11  import java.util.Random;
12  import java.util.concurrent.TimeUnit;
13
14  @Fork(value = 1)
15  @OutputTimeUnit(TimeUnit.Ticks)
16  @State(Scope.Benchmark)
17  public class TimeMeasureLab4 {
18    @Param({"100", "200", "300" , "400", "500", "600", "700
         ", "800", "900", "1000"  })
19    public int params;
20
21    public  int[][] generate_matrix(int n, int m) {
22      int[][] new_matrix = new int[n][m];
23      Random random = new Random();
24      for (int i = 0; i < n; i++) {
25        for (int j = 0; j < m; j++) {
```

```
26              new_matrix[i][j] = random.nextInt(1000);
27            }
28          }
29          return new_matrix;
30        }
31
32
33        public static void main(String[] args) throws
              IOException, RunnerException {
34          Options options = new OptionsBuilder().include(
                TimeMeasureLab4.class.getSimpleName()).
35          forks(1).resultFormat(ResultFormatType.LATEX).result(
                "MatrixComparison.tex").build();
36          new Runner(options).run();
37
38        }
39        @Benchmark
40        @BenchmarkMode(Mode.AverageTime)
41        @Warmup(iterations = 2)
42        @Measurement(iterations = 10)
43        public int[][] MeasureStandart() {
44          int[][] mart1 = generate_matrix(params, params);
45          int[][] matr2 = generate_matrix(params, params);
46          return Matrix.MultVinOpt(mart1, matr2);
47
48        }
49
50        @Benchmark
51        @BenchmarkMode(Mode.AverageTime)
52        @Warmup(iterations = 2)
53        @Measurement(iterations = 10)
54        public int[][] MeasureParallel2() throws
              InterruptedException {
55          int[][] mart1 = generate_matrix(params, params);
56          int[][] matr2 = generate_matrix(params, params);
57          return MatrixParallel.MultVinOpt(mart1, matr2, 2);
58        }
59
60        @Benchmark
61        @BenchmarkMode(Mode.AverageTime)
```

```
62      @Warmup(iterations = 2)
63      @Measurement(iterations = 10)
64      public int[][] MeasureParallel4() throws
            InterruptedException {
65        int[][] mart1 = generate_matrix(params, params);
66        int[][] matr2 = generate_matrix(params, params);
67        return MatrixParallel.MultVinOpt(mart1, matr2, 4);
68      }
69
70      @Benchmark
71      @BenchmarkMode(Mode.AverageTime)
72      @Warmup(iterations = 2)
73      @Measurement(iterations = 10)
74      public int[][] MeasureParallel8() throws
            InterruptedException {
75        int[][] mart1 = generate_matrix(params, params);
76        int[][] matr2 = generate_matrix(params, params);
77        return MatrixParallel.MultVinOpt(mart1, matr2, 8);
78      }
79
80      @Benchmark
81      @BenchmarkMode(Mode.AverageTime)
82      @Warmup(iterations = 2)
83      @Measurement(iterations = 10)
84      public int[][] MeasureParallel1() throws
            InterruptedException {
85        int[][] mart1 = generate_matrix(params, params);
86        int[][] matr2 = generate_matrix(params, params);
87        return MatrixParallel.MultVinOpt(mart1, matr2, 1);
88      }
89
90    }
```

## 3.4 Тестирование

Тестирование программы производилось с помощью библиотеки Junit 5.18 методом черного ящика. Код тестирования представлен в листинге ??

Всего было реализованно 7 тестовых случаев:

- некорректный размер матриц. Алгоритм должен возвращать Null;
- размер матриц равен 2;
- матрицы содержит отрицательные элементы;
- корректность работы стандартной реализации с Виноградом;
- корректность работы стандартной реализации с оптимизированным Виноградом на случайных значениях.

```
1  package lab4Tests;
2  import org.junit.jupiter.api.Assertions;
3  import org.junit.jupiter.api.Test;
4  import lab4.Matrix;
5  import lab4.MatrixParallel;
6
7  public class Tests {
8
9    int[][] A = new int[2][2];
10   int[][] B = new int[2][2];
11   int[][] C = new int[0][2];
12   int[][] D = new int[2][0];
13   int[][] E = new int[5][5];
14
15
16
17   @Test
18   public void  testVinOpt() {
19     for (int i = 0, counter = 1; i < 2; i++) {
20       for (int j = 0; j < 2;  j++) {
21         A[i][j] = counter;
22         B[i][j] = counter++;
23       }
24     }
25     int[][] res = {{7,10}, {15,22}};
26     Assertions.assertArrayEquals(res, Matrix.MultVinOpt(A
         , B));
27
```

```
28        for (int i = 0, counter = -1; i < 2; i++) {
29          for (int j = 0; j < 2;  j++) {
30            A[i][j] = counter;
31            B[i][j] = counter--;
32          }
33        }
34        Assertions.assertArrayEquals(res, Matrix.MultVinOpt(A
              , B));
35        Assertions.assertArrayEquals(null, Matrix.MultVinOpt(
              A, C));
36        Assertions.assertArrayEquals(null, Matrix.MultVinOpt(
              A,D));
37        Assertions.assertArrayEquals(null, Matrix.MultVinOpt(
              A, E));
38      }
39
40      @Test
41      public void TestVinParallel() throws
           InterruptedException {
42        for (int i = 0, counter = 1; i < 2; i++) {
43          for (int j = 0; j < 2;  j++) {
44            A[i][j] = counter;
45            B[i][j] = counter++;
46          }
47        }
48        int[][] res = {{7,10}, {15,22}};
49        Assertions.assertArrayEquals(res, MatrixParallel.
              MultVinOpt(A, B, 2));
50
51        for (int i = 0, counter = -1; i < 2; i++) {
52          for (int j = 0; j < 2;  j++) {
53            A[i][j] = counter;
54            B[i][j] = counter--;
55          }
56        }
57        Assertions.assertArrayEquals(res, MatrixParallel.
              MultVinOpt(A, B,2));
58        Assertions.assertArrayEquals(null, MatrixParallel.
              MultVinOpt(A, C, 2));
59        Assertions.assertArrayEquals(null, MatrixParallel.
```

```
            MultVinOpt(A,D, 2));
60          Assertions.assertArrayEquals(null, MatrixParallel.
            MultVinOpt(A, E, 2));
61      }
62  }
```

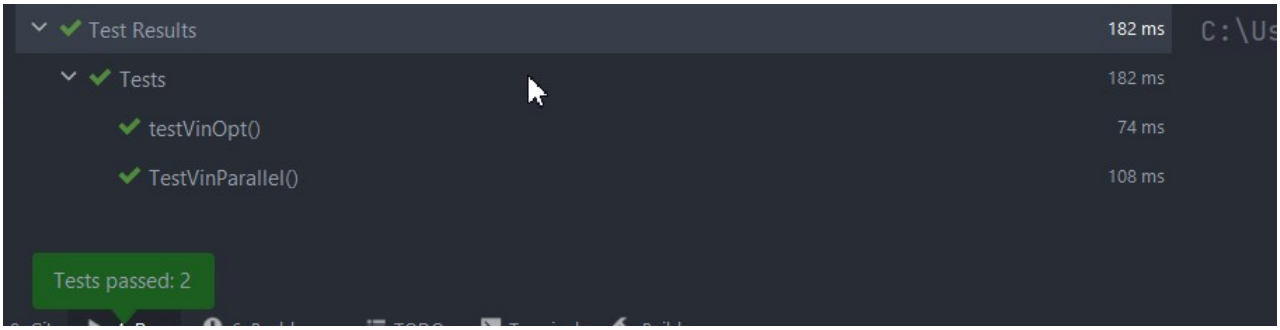


Рис. 3.1:

Результаты тестирование представлены на рисунке 3.1

## 3.5 Вывод

В данном разделе были рассмотрены основные сведения о модулях программы, листинг кода.

# 4. Исследовательская часть

## 4.1 Технические характеристики ЭВМ на которой проводились эксперименты

- операционная система - Windows 10 pro Версия 1909;
- процессор - intel core i5-8520U ядер:4, логических процессоров:8;
- оперативная память - 8 Гб.

## 4.2 Сравнительный анализ на основе замеров времени

Для замера времени использовалась платформа Java MicroBencmark Harness.[4] Для точности результат брался по среднему времени для 10 экспериментов, 2 дополнительных эксперимента не учитывается и служат для разгона JVM. Java MicroBencmark Harness снижает влияние сторонних процессов, следовательно требуется меньшее число замеров.

В рамках данного эксперимента было выстроена зависимость времени работы алгоритма умножения Винограда без создания дополнительных потоков, с созданием 1,2,4, 8,16,32,64. На предоставленном ниже графике (Рис. 4.1).

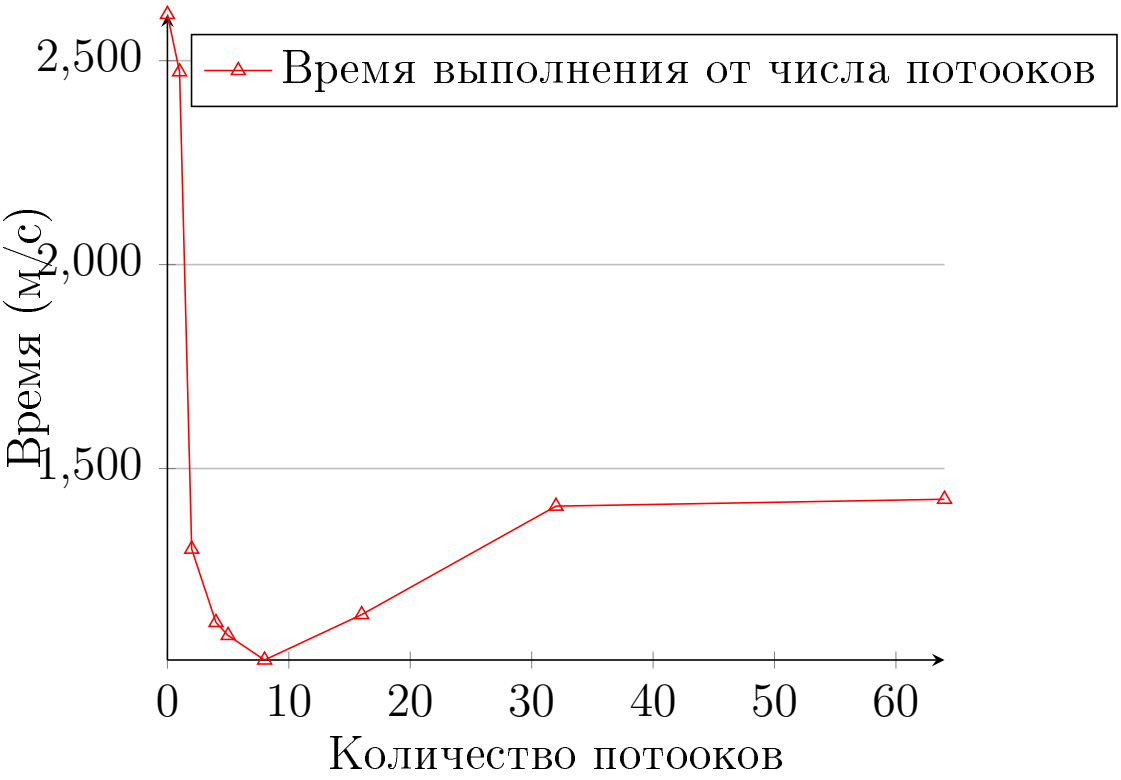


Рис. 4.1: Сравнение времени работы для разного количество дополнительно созданным потоков

## 4.3 Вывод

По результатам исследования видно, что самым оптимальным количеством потоков является число 8. Далее производительность начинает ухудшаться. Экспериментальные данные совпадают с ожидаемыми - количество логических процессоров 8, поэтому каждый из 8 потоков, ждет своего кванта, намного меньшее время, чем в программе на 16 потоках Алгоритм на 8 потоках, работает в два с половиной раза быстрее алгоритма, в котором не создается дополнительных потоков.

# Заключение

Цель работы достигнута, все задачи выполнены.

1. был изучен принцип работы параллельных вычислений;
2. реализован алгоритма винограда с возможностью распараллеливания вычислений;
3. все алгоритмы протестированы методом черного ящика;
4. экспериментально было подтверждено различие во временно́й эффективности алгоритмов, задействующих разное количество потоков;

# Литература


[1] Le Gall, F. (2012), "Faster algorithms for rectangular matrix multiplication Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012), pp. 514–523

[2] И. В. Белоусов(2006), Матрицы и определители, учебное пособие по линейной алгебре, с. 1 - 16

[3] Воеводин В. В., Воеводин Вл. В. Параллельные вычисления. — СПб: БХВ-Петербург, 2002. — 608 с.

[4] Micro Benchmarking with Java[Электронный ресурс] - режим доступа: https://www.baeldung.com/java-microbenchmark-harness