МГТУ им. Баумана

Лабораторная работа №4

По курсу: "Анализ алгоритмов"

# Параллельное программирование

Студент: Аникин И. А., ИУ7-51Б

Преподаватель: Волкова Л.Л.

*Москва, 2020*

# Содержание

# Введение

Когда появились первые процессоры, они были однозадачными, то есть одновременно выполняли только одну команду.

На сегодняшний день процессоры состоят из нескольких ядер. Нестрого говоря, ядро - это самостоятельный процессор. То есть четырехядерный процессор это четыре самостоятельных процессора в едином корпусе. Каждое ядро может выполнять одну команду независимо от работы других ядер. Существует технологии, благодаря которым одно ядро может выполнять параллельно два потока команд.
Параллельное программирование имеет ряд преимуществ по сравнению с однопоточным программированием:

- улучшает время работы и вычислений;
- позволяет обеспечивать одновременное выполнение различных функций системы.

# 1. Аналитическая часть

Рассмотрим цель и задачи данной лабораторной работы.

Цель лабораторной работы - разработать и провести сравнительный анализ однопоточного и многопоточных алгоритмов.
Для достижения данной цели можно выделить следующие задачи:

- разработать однопоточный алгоритм Винограда и две многопоточные версии этого алгоритма;
- реализовать разработанные алгоритмы;
- провести тестирование разработанных алгоритмов по методу черного ящика;
- провести замеры времени работы реализаций алгоритмов при разном числе потоков;
- провести сравнительный анализ временной эффективности разработанных алгоритмов.

Алгоритм Винограда - улучшенный алгоритм умножения матриц, который уменьшает свою вычислительную сложность за счет оптимизации вычисления произведения строки и столбца.
Пусть $u = (u_1, u_2, u_3, u_4)$ строка матрицы A, а $v = (v_1, v_2, v_3, v_4)$ столбец матрицы B.
В данном алгоритме вместо классического умножения $u_1v_1 + u_2v_2 + u_3v_3 + u_4v_4$ произведение вычисляется следующим образом:
$(u_1 + v_2)(u_2 + v_1) + (u_3 + v_4)(u_4 + v_3) - u_1u_2 - u_3u_4 - v_1v_2 - v_3v_4$
Если количество столбцов нечетно, то это является худшим случаем и к произведению добавляется множитель $+u_5v_5$.

Такой способ содержит меньшее количество умножений, а значит является более быстрым при вычислении на ЭВМ.

# 2. Конструкторская часть

## 2.1. Требования к ПО

Разработанное ПО должно обладать следующей функциональностью:

- считывание двух матриц;
- выбор алгоритма умножения;
- вывод результата умножения;
- осуществление замеров процессорного времени работы реализаций алгоритмов.

## 2.2. Разработка алгоритмов

В ходе данной лабораторной работы будет реализовано 3 различных алгоритма умножения матриц:

- однопоточный алгоритм Винограда;
- многопоточный алгоритм Винограда с параллельным вычислением массивов multA, multB и умножением;
- многопоточный алгоритм Винограда с последовательным вычислением multA, multB и параллельным умножением.

На рисунке 2 в приложении 1 представлены схемы потоков для вычисления массивов multA, multB и умножения.

На рисунке 3 в приложении 2 представлена схема однопоточного алгоритма Винограда.

На рисунках 4-5 в приложении 3 представлена схема многопоточного алгоритма Винограда с параллельным вычислением массивов multA, multB и умножением.

На рисунках 6 в приложении 4 представлена схема многопоточного алгоритма Винограда с последовательным вычислением массивов multA, multB и параллельным умножением.

На рисунке 7 в приложении 5 представлена схема многопоточного алгоритма Винограда с параллельным вычислением массивов multA, multB и умножением.

На рисунке 8 в приложении 6 представлена схема многопоточного алгоритма Винограда с последовательным вычислением массивов multA, multB и параллельным умножением.

На рисунках 7-8 блок I - заполнение массива multA, блок II - заполнение массива multB, блок III - умножение, пустой блок - точка синхронизации потоков.

# 3. Технологическая часть

В ходе выполнения лабораторной работы были реализованы следующие модули:

- классы multAThread, multBThread, в которых реализованы потоки заполнения массивов multA, multB;
- класс winoThread, в котором реализован поток умножения матриц;
- класс Testing, предназначенный для тестирования реализованных алгоритмов;
- функция analysis, предназначенная для замеров времени выполнения алгоритмов при различном числе потоков;
- функция main, предоставляющая пользователю возможности умножения матриц выбранным методом.

В качестве языка разработки был выбран язык java, т. к. он имеет ряд преимуществ для достижения цели лабораторной работы:

- высокая скорость разработки программ;
- C-подобный синтаксис языка;
- наличие в стандартной библиотеке класса для работы с потоками, что позволяет легко реализовывать многопоточные алгоритмы.

## 3.1. Листинги алгоритмов

На листинге 1 приведен однопоточный алгоритм Винограда.

Листинг 1: однопоточный алгоритм Винограда

```
public static void singleWinograd(int[][] a, int[][] b, int[][] c) {
        int m = a.length;
        int n = b.length;
        int q = b[0].length;

        int[] multA = new int[m];
        for (int i = 0; i < m; i++) {
            multA[i] = 0;
            for (int k = 0; k < n - n%2; k += 2)
                multA[i] -= a[i][k] * a[i][k + 1];
        }

        int[] multB = new int[q];
        for (int i = 0; i < q; i++) {
            multB[i] = 0;
            for (int k = 0; k < n - n%2; k += 2)
                multB[i] -= b[k][i] * b[k + 1][i];
        }

        for (int i = 0; i < m; i++)
            for (int j = 0; j < q; j++) {
                c[i][j] = multA[i] + multB[j];
                for (int k = 0; k < n - n%2; k += 2)
                    c[i][j] += (a[i][k] + b[k+1][j]) * (a[i][k+1] + b
                        [k][j]);
                if (n % 2 == 1)
                    c[i][j] += a[i][n-1] * b[n-1][j];
            }
    }
```

На листинге 2 приведены классы потоков для заполнения массивов и умножения матриц.

Листинг 2: Классы потоков

```
class multAThread extends Thread {
    int[][] matrix;
    int[] array;
    int begin;
    int end;

    public multAThread(int[][] matrix, int[] array, int begin, int
       end) {
        this.matrix = matrix;
```

```
9           this.array = array;
10          this.begin = begin;
11          this.end = end;
12      }
13
14      @Override
15      public void run() {
16          int m = matrix.length;
17          int n = matrix[0].length;
18          for (int i = begin; i < end; i++) {
19              array[i] = 0;
20              for (int k = 0; k < n - n%2; k += 2)
21                  array[i] -= matrix[i][k] * matrix[i][k + 1];
22          }
23          super.run();
24      }
25  }
26
27  class multBThread extends Thread {
28      int[][] matrix;
29      int[] array;
30      int begin;
31      int end;
32
33      public multBThread(int[][] matrix, int[] array, int begin, int
            end) {
34          this.matrix = matrix;
35          this.array = array;
36          this.begin = begin;
37          this.end = end;
38      }
39
40      @Override
41      public void run() {
42          int q = matrix[0].length;
43          int n = matrix.length;
44          for (int i = begin; i < end; i++) {
45              array[i] = 0;
46              for (int k = 0; k < n - n%2; k += 2)
47                  array[i] -= matrix[k][i] * matrix[k + 1][i];
48          }
49          super.run();
50      }
51  }
52
53  class winoThread extends Thread {
54      int[][] a;
55      int[][] b;
56      int[][] c;
57      int[] multA;
```

```
58      int[] multB;
59      int begin;
60      int end;
61
62      public winoThread(int[][] a, int[][] b, int[][] c, int[] multA,
            int[] multB, int begin, int end) {
63          this.a = a;
64          this.b = b;
65          this.c = c;
66          this.multA = multA;
67          this.multB = multB;
68          this.begin = begin;
69          this.end = end;
70      }
71
72      @Override
73      public void run() {
74          int m = a.length;
75          int n = b.length;
76          int q = b[0].length;
77
78          for (int i = begin; i < end; i++)
79              for (int j = 0; j < q; j++) {
80                  c[i][j] = multA[i] + multB[j];
81                  for (int k = 0; k < n - n%2; k += 2)
82                      c[i][j] += (a[i][k] + b[k+1][j]) * (a[i][k+1] + b
                            [k][j]);
83                  if (n%2 == 1)
84                      c[i][j] += a[i][n-1] * b[n-1][j];
85              }
86          super.run();
87      }
88  }
```

На листинге 3 приведен многопоточный алгоритм Винограда с параллельным заполнением массивов и умножением.

Листинг 3: Многопоточный алгоритм Винограда 1

```
1  public static void multiWinograd(int[][] a, int[][] b, int[][] c, int
       numThreads) throws InterruptedException {
2          int m = a.length;
3          int n = b.length;
4          int q = b[0].length;
5
6          int[] multA = new int[m];
7          int[] multB = new int[q];
8          Thread[] threads = new Thread[numThreads];
9          int threadsInUse = 0;
10
```

```
11          int rowsPerThread = m / numThreads;
12          if (rowsPerThread == 0)
13              rowsPerThread = 1;
14
15          for (int i = 0, row = 0; i < numThreads && row < m; i++, row
              += rowsPerThread) {
16              if (i+1 != numThreads)
17                  threads[i] = new multAThread(a, multA, row, row +
                      rowsPerThread);
18              else
19                  threads[i] = new multAThread(a, multA, row, m);
20              threadsInUse++;
21          }
22
23          for (int i = 0; i < threadsInUse; i++)
24              threads[i].start();
25
26          for (int i = 0; i < threadsInUse; i++)
27              threads[i].join();
28
29          threadsInUse = 0;
30
31          int colsPerThread = q / numThreads;
32          if (colsPerThread == 0)
33              colsPerThread = 1;
34
35          for (int i = 0, col = 0; i < numThreads && col < q; i++, col
              += colsPerThread) {
36              if (i+1 != numThreads)
37                  threads[i] = new multBThread(b, multB, col, col +
                      colsPerThread);
38              else
39                  threads[i] = new multBThread(b, multB, col, q);
40              threadsInUse++;
41          }
42
43          for (int i = 0; i < threadsInUse; i++)
44              threads[i].start();
45
46          for (int i = 0; i < threadsInUse; i++)
47              threads[i].join();
48
49          threadsInUse = 0;
50          rowsPerThread = m / numThreads;
51          if (rowsPerThread == 0)
52              rowsPerThread = 1;
53          for (int i = 0, row = 0; i < numThreads && row < m; i++, row
              += rowsPerThread) {
54              if (i+1 != numThreads)
55                  threads[i] = new winoThread(a, b, c, multA, multB,
```

```
                    row, row + rowsPerThread);
56              else
57                  threads[i] = new winoThread(a, b, c, multA, multB,
                        row, m);
58              threadsInUse++;
59          }
60          for (int i = 0; i < threadsInUse; i++)
61              threads[i].start();
62
63          for (int i = 0; i < threadsInUse; i++)
64              threads[i].join();
65      }
```

На листинге 4 приведен многопоточный алгоритм Винограда с последовательным заполнением массивов и параллельным умножением.

Листинг 4: Многопоточный алгоритм Винограда 2

```
1  public static void multiWinograd2(int[][] a, int[][] b, int[][] c,
       int numThreads) throws InterruptedException {
2          int m = a.length;
3          int n = b.length;
4          int q = b[0].length;
5
6          int[] multA = new int[m];
7          for (int i = 0; i < m; i++) {
8              multA[i] = 0;
9              for (int k = 0; k < n - n%2; k += 2)
10                 multA[i] -= a[i][k] * a[i][k + 1];
11         }
12
13         int[] multB = new int[q];
14         for (int i = 0; i < q; i++) {
15             multB[i] = 0;
16             for (int k = 0; k < n - n%2; k += 2)
17                 multB[i] -= b[k][i] * b[k + 1][i];
18         }
19
20         Thread[] threads = new Thread[numThreads];
21         int threadsInUse = 0;
22
23         int rowsPerThread = m / numThreads;
24         if (rowsPerThread == 0)
25             rowsPerThread = 1;
26         for (int i = 0, row = 0; i < numThreads && row < m; i++, row
               += rowsPerThread) {
27             if (i+1 != numThreads)
28                 threads[i] = new winoThread(a, b, c, multA, multB,
                        row, row + rowsPerThread);
29             else
```

```
30                  threads[i] = new winoThread(a, b, c, multA, multB,
                        row, m);
31              threadsInUse++;
32          }
33
34          for (int i = 0; i < threadsInUse; i++)
35              threads[i].start();
36
37          for (int i = 0; i < threadsInUse; i++)
38              threads[i].join();
39      }
```

## 3.2. Тестирование

Тестирование разработанных алгоритмов проводилось на наборе тестов, который представлен в таблице 1.

Таблица 1: Набор тестов

| Матрица A | Матрица B | Ожидаемые выходные данные |
|---|---|---|
| $\begin{matrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{matrix}$ | $\begin{matrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{matrix}$ | $\begin{matrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{matrix}$ |
| $\begin{matrix}1 & 2 & 3\\ 3 & 2 & 1\\ 2 & 3 & 1\end{matrix}$ | $\begin{matrix}1 & 1 & 1\\ 2 & 2 & 2\\ 3 & 3 & 3\end{matrix}$ | $\begin{matrix}14 & 14 & 14\\ 10 & 10 & 10\\ 11 & 11 & 11\end{matrix}$ |
| $\begin{matrix}3 & 7 & -2\\ -11 & 7 & 3\end{matrix}$ | $\begin{matrix}1 & 0 & 11\\ -1 & 7 & 17\\ 2 & 6 & -14\end{matrix}$ | $\begin{matrix}-8 & 37 & 180\\ -12 & 67 & -44\end{matrix}$ |
| $\begin{matrix}3 & 7 & -2 & 0\\ -11 & 7 & 3 & 1\end{matrix}$ | $\begin{matrix}1 & 0 & 11\\ -1 & 7 & 17\\ 2 & 6 & -14\\ -3 & -4 & -22\end{matrix}$ | $\begin{matrix}-8 & 37 & 180\\ -15 & 63 & -66\end{matrix}$ |

Все тесты пройдены успешно.

# 4. Исследовательская часть

## 4.1. Сравнение временных характеристик алгоритмов

Проведем исследование временных характеристик реализованных алгоритмов. Замеры времени происходили с использованием ЦП Intel Xeon E3-1270 V2 и 16 ГБ ОЗУ. Число логических процессоров данного ЦП - 8.
Ниже представлены графики зависимости времени работы алгоритмов при различном числе потоков. На вход подаются две матрицы.
На рисунке 1 изображено время работы с количеством потоков от 1 до $4n$, где $n$ - реальное число потоков процессора.

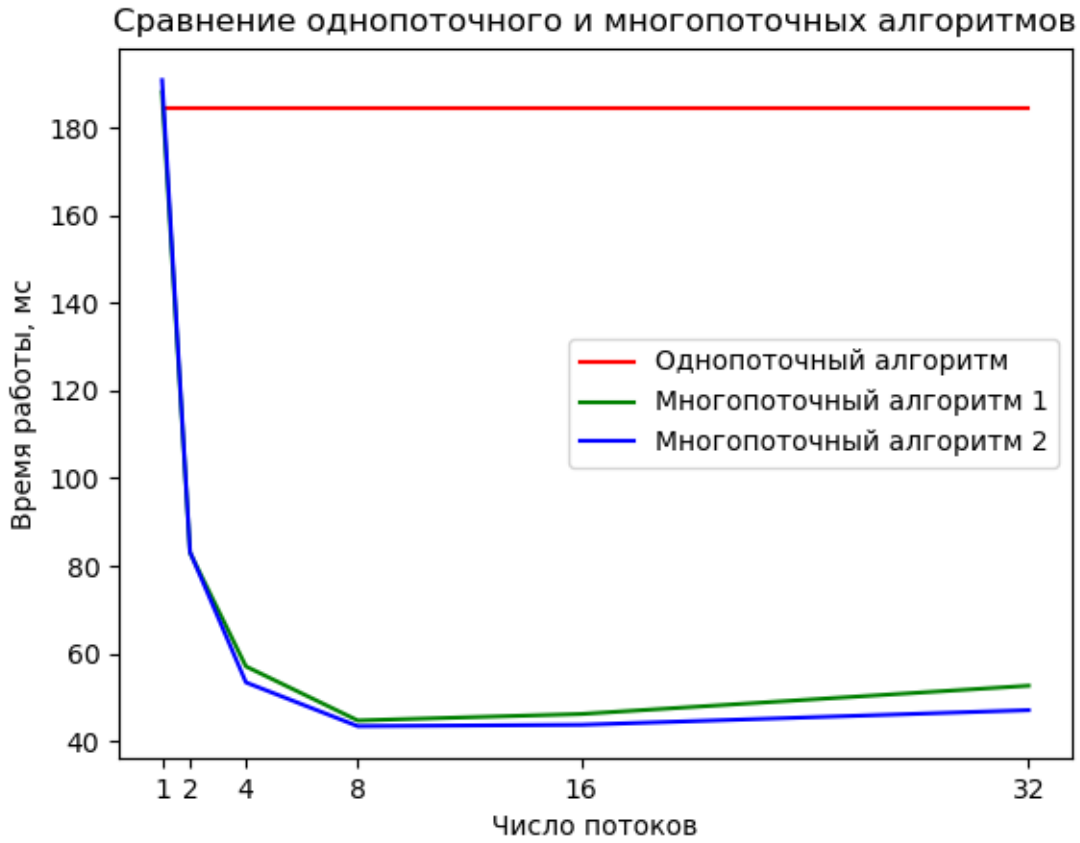


Рис. 1: Зависимость времени работы от числа потоков

Исходя из данных графиков, видно, что наиболее быстрым является алгоритм с последовательным заполнением массивов и параллельным умножением. А значит, можно сделать вывод о том, что параллельное заполнение массивов multA и multB не дает существенного выигрыша во времени работы, а с учетом затрат на работу с потоками даже ухудшает его.

К применению рекомендуется алгоритм с последовательным заполнением массивов multA, multB и параллельным умножением как наименее затратный по времени.

Также видно, что наиболее быстрое время работы алгоритма достигается, когда количество потоков, создаваемых алгоритмом, равно количеству логических процессоров. Дальнейшее увеличение числа потоков не дает выигрыша во времени, так как потоки начинают конфликтовать за процессорное время и простаивать[1].

# Заключение

В ходе выполнения лабораторной работы были выполнены следующие задачи:

- разработаны однопоточный алгоритм Винограда и две многопоточные версии этого алгоритма;
- реализованы разработанные алгоритмы;
- проведено тестирование разработанных алгоритмов по методу черного ящика;
- проведены замеры времени работы реализаций алгоритмов при разном числе потоков;
- проведен сравнительный анализ временной эффективности разработанных алгоритмов.

Цель лабораторной работы достигнута, все задачи выполнены.

Даны рекомендации по использованию исследованных алгоритмов.

По итогам достигнутых задач можно сделать следующие выводы.
К применению рекомендуется алгоритм с последовательным заполнением массивов и параллельным умножением, т. к. он . А значит можно сделать вывод о том, что параллельное заполнение массивов multA и multB не дает существенного выигрыша во времени работы, а с учетом затрат на работу с потоками даже ухудшает его.

Также видно, что наиболее быстрое время работы алгоритма достигается, когда количество потоков, создаваемых алгоритмом, равно количеству логических процессоров.

# Список литературы

[1] Миллер, Р. Последовательные и параллельные алгоритмы: Общий подход / Р. Миллер, Л. Боксер ; пер. с англ. — М. : БИНОМ. Лаборатория знаний, 2006. — 406 с.

# Приложения

Приложение 1

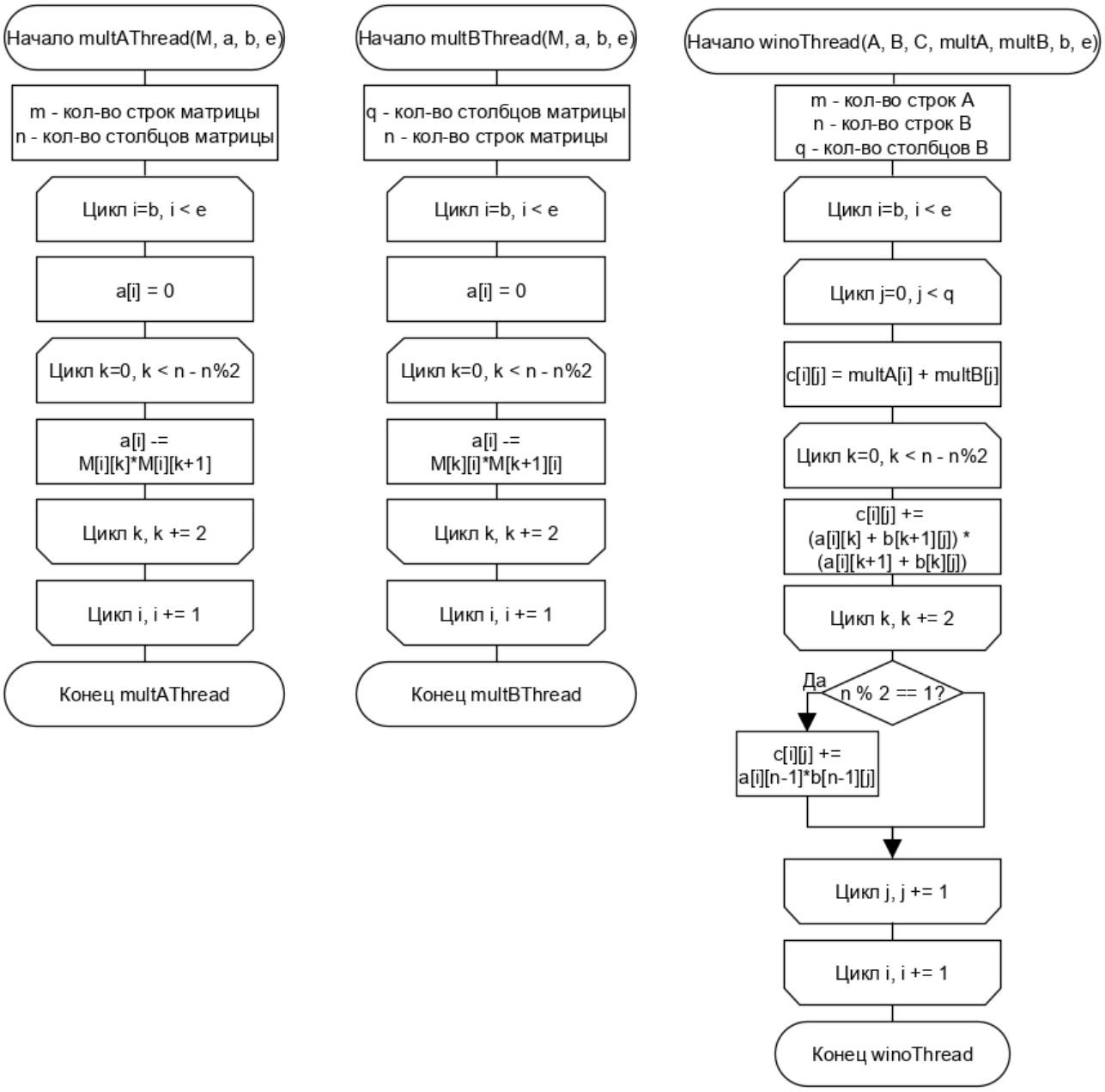


Рис. 2: Схемы потоков

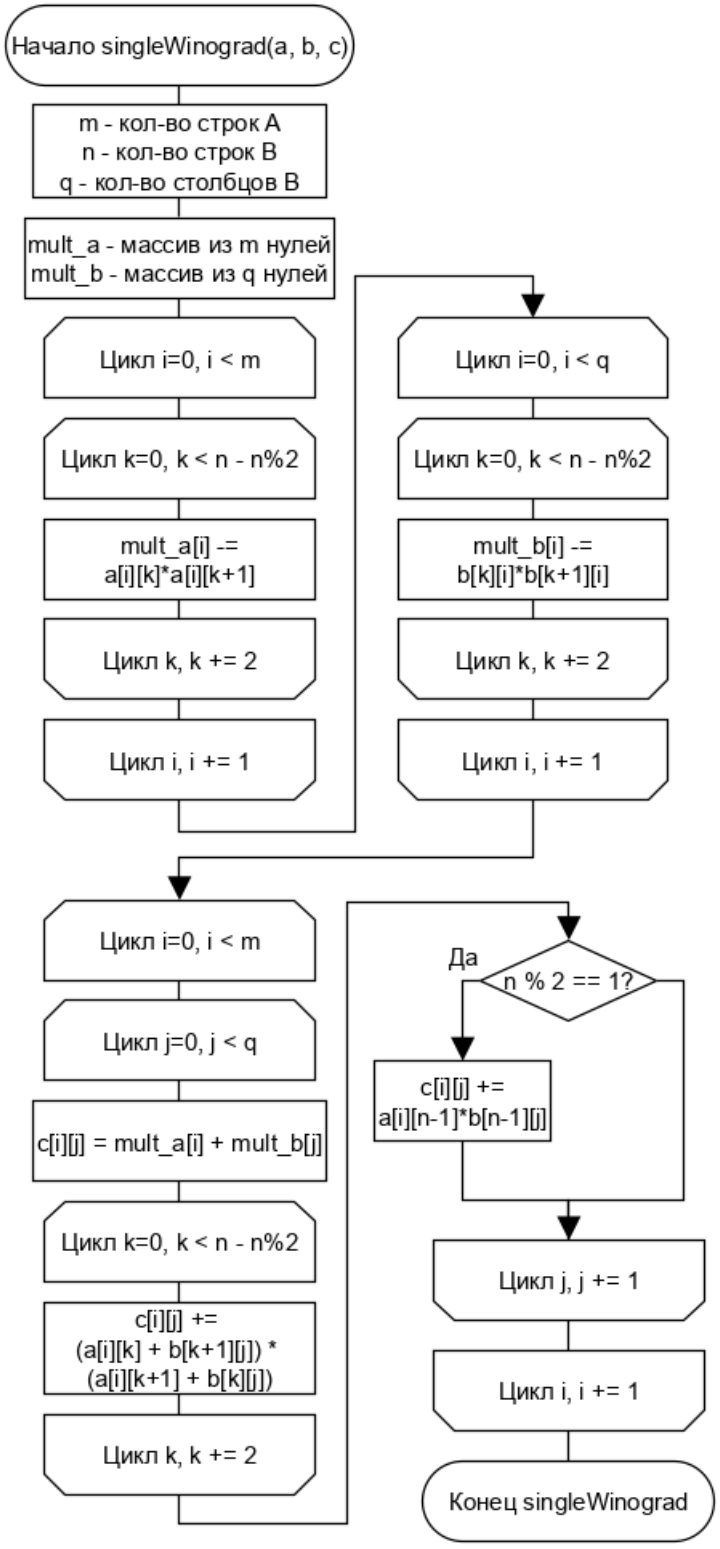


Рис. 3: Схема одногопоточного алгоритма Винограда

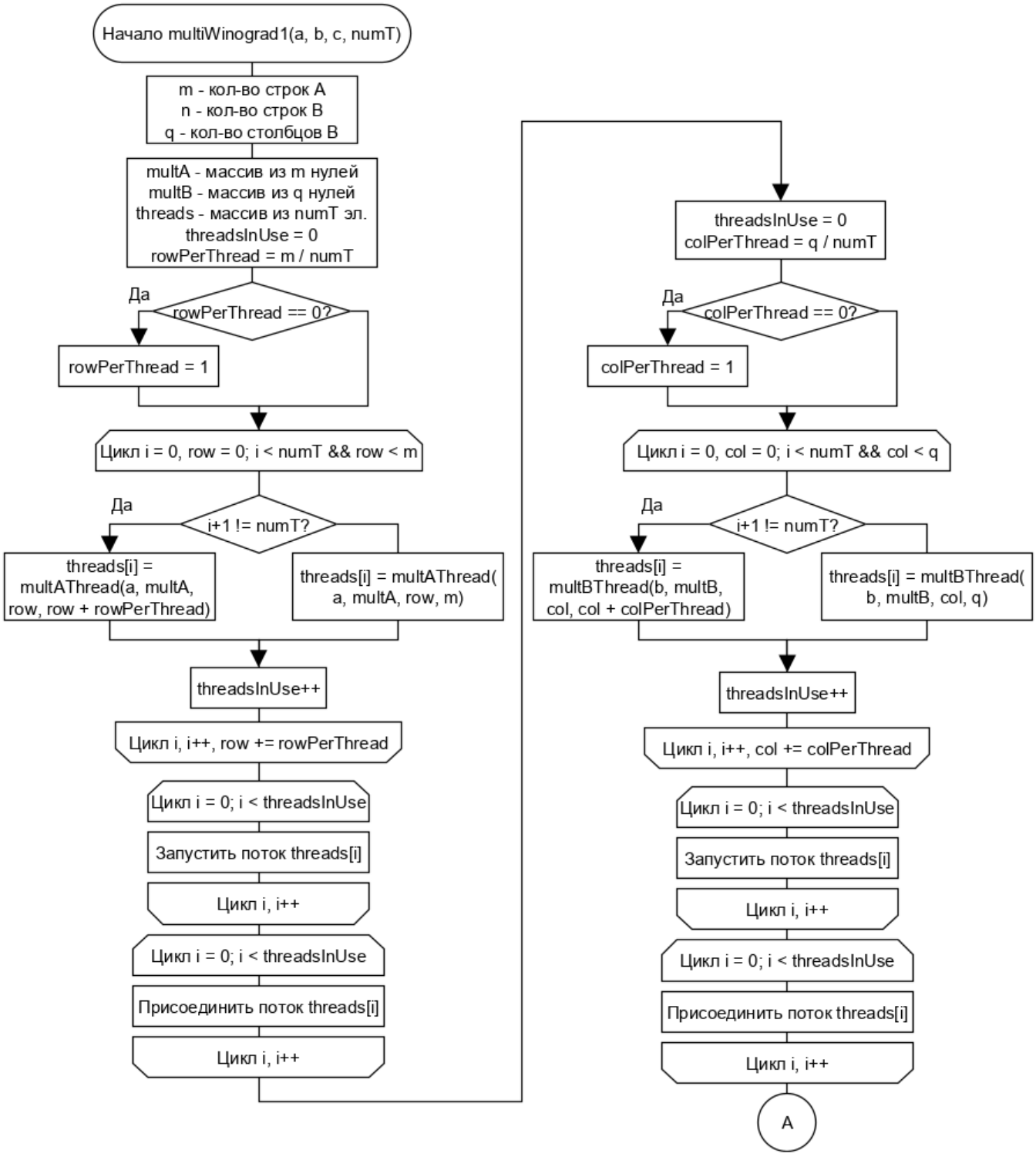


Рис. 4: Схема многопоточного алгоритма Винограда с параллельным заполнением

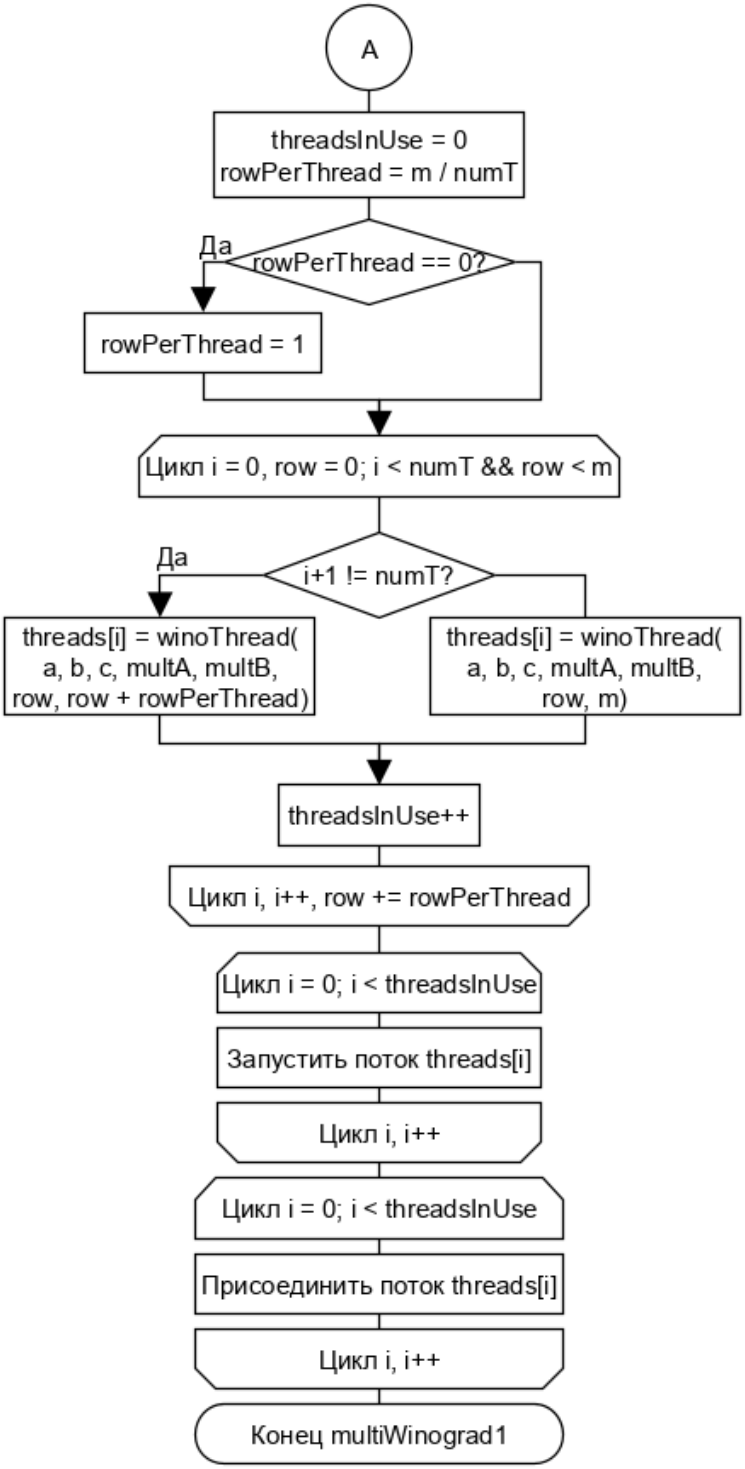


Рис. 5: Схема многопоточного алгоритма Винограда с параллельным заполнением (продолжение)

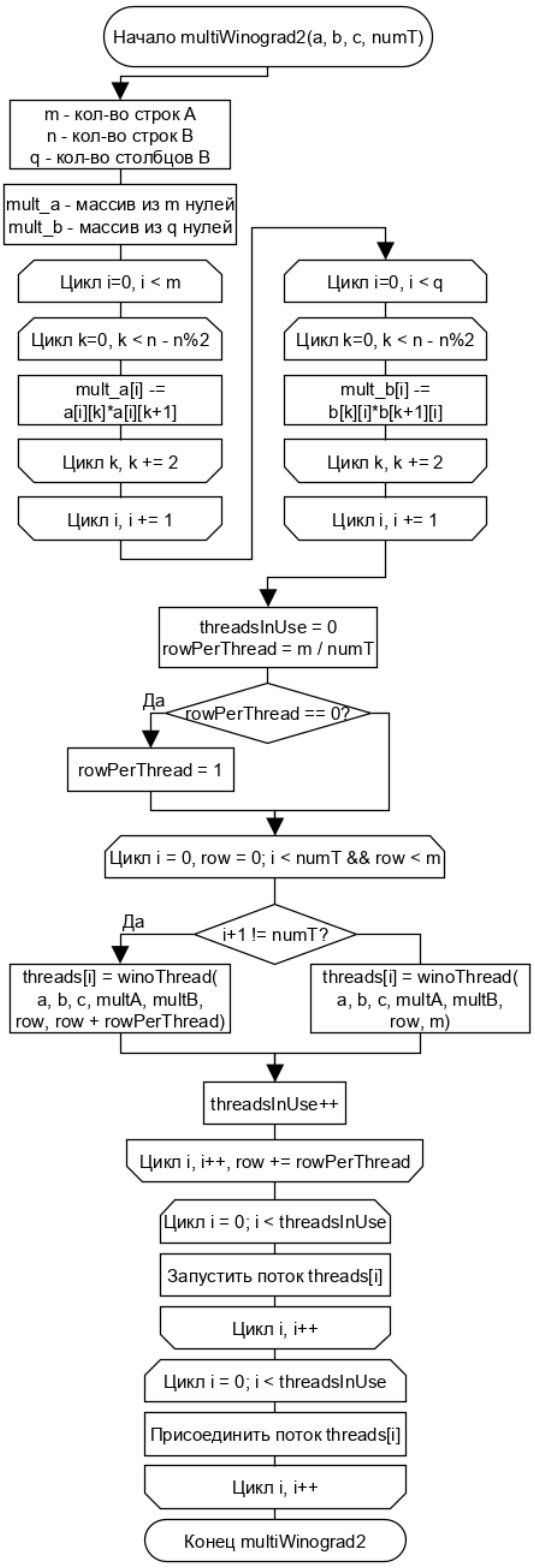


Рис. 6: Схема многопоточного алгоритма Винограда с последовательным заполнением

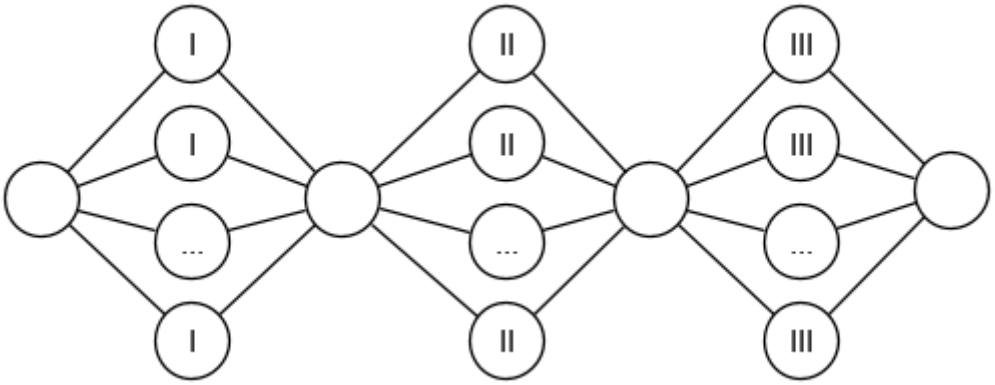


Рис. 7: Схема многопоточного алгоритма Винограда 1

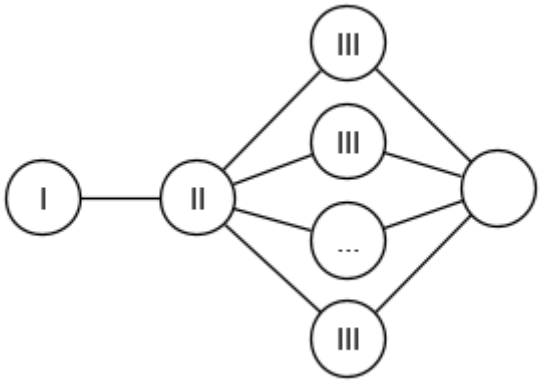


Рис. 8: Схема многопоточного алгоритма Винограда 2