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 №4

По дисциплине: Анализ Алгоритмов

Тема: Распараллеливание потоков

Студент Чаушев А.К..

Группа ИУ7-56Б

Оценка (баллы) _____

Преподаватели Волкова Л.Л., Строганов Ю.В.

Москва — 2020 г.

Введение

Матрица [1] – математический объект, эквивалентный двумерному массиву. Числа располагаются в матрице по строкам и столбцам. Умножение матриц имеет многочисленные применения в математике, физике, программировании.

Целью данной лабораторной работе является изучение и реализация алгоритма Винограда для умножения матриц с использованием потоков.

В данной лабораторной работе ставятся следующие задачи:

1) изучение распараллеливания вычислений и работа с потоками;
2) реализация распараллеленных вычислений;
3) экспериментальное сравнение работы алгоритма на разном количестве потоков.

1 Аналитическая часть

В данной части будут рассмотрены основные теоретические аспекты, связанные с парарельное умножения матриц. Рассмотрим как можно решить эту задачу.

1.1 Постановка задачи

Пусть даны две прямоугольные матрицы A и B размерности $l \times m$ и $m \times n$ соответственно, указанные в формуле 1.

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

Тогда матрица C будет размерностью $l \times n$ в формуле 2, в которой каждый элемент равен выражению из формулы 3 [1].

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \cdots & c_{ln} \end{bmatrix} \tag{2}$$

$$c_{ij} = \sum_{k=1}^{m} a_{ik} \cdot b_{kj}, i = \overline{1;l}, j = \overline{1;n} \tag{3}$$

Операция умножения двух матриц выполнима только в том случае, если число столбцов в первом сомножителе равно числу строк во втором; в этом случае говорят, что матрицы согласованы. В частности, умножение всегда выполнимо, если оба сомножителя — квадратные матрицы одного и того же порядка [1].

Таким образом, из существования произведения $A \times B$ вовсе не следует существование произведения $B \times A$.

Помимо обычного перемножения матриц по формуле существуют модификации, работающие быстрее. Рассмотрим в данной лабораторной работе алгоритм Винограда, являющийся одним из самых эффективных по времени алгоритмов умножения матриц. Этот алгоритм основывается на подготовке вычислений перед вычислением результирующей матрицы. Если разложить формулу 3 на суммы, то получается результат, видимый в формуле 4.

$$c_{ij} = \sum_{k=1}^{\frac{n}{2}} (a_{i,2k-1} + b_{2k,j}) \cdot (a_{i,2k} + b_{2k-1,j}) - \underbrace{\sum_{k=1}^{\frac{n}{2}} a_{i,2k-1} \cdot a_{i,2k} - \sum_{k=1}^{\frac{n}{2}} b_{2k-1,j} \cdot b_{2k,j}}_{\text{Можно вычислить заранее}} \tag{4}$$

Таким образом, можно заранее вычислить две последние суммы, поскольку они вычисляются многократно для каждой строки в одном столбце в случае первой и для каждого столбца из одной строки в случае второй сумм, что уменьшает долю умножения. Также можно заметить, что вычисление каждого нового элемента результирующей матрицы не влияет на вычисление следующих, то есть каждый элемент матрицы считается отдельно. По этой причине можно проделать действия по распараллеливанию вычислений и тем самым увеличить скорость расчета результата.

1.2 Выводы

Умножение матриц – необходимый инструмент, для которого есть пути ускорения вычислений за счет уменьшения доли умножения и распараллеливания вычислений.

2 Конструкторская часть

Рассмотрим алгоритм Винограда и способы его расспаралеливания.

2.1 Схемы алгоритмов

На рисунке 1 – 2 изображена схема алгоритма Винограда.

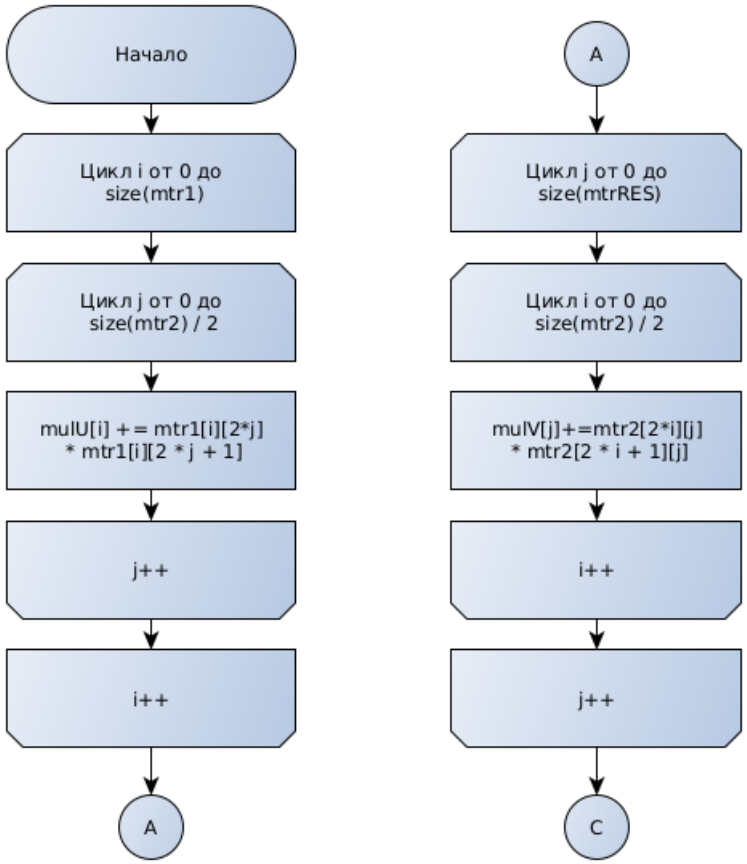

Рис. 1 – Схема алгоритма Винограда часть 1

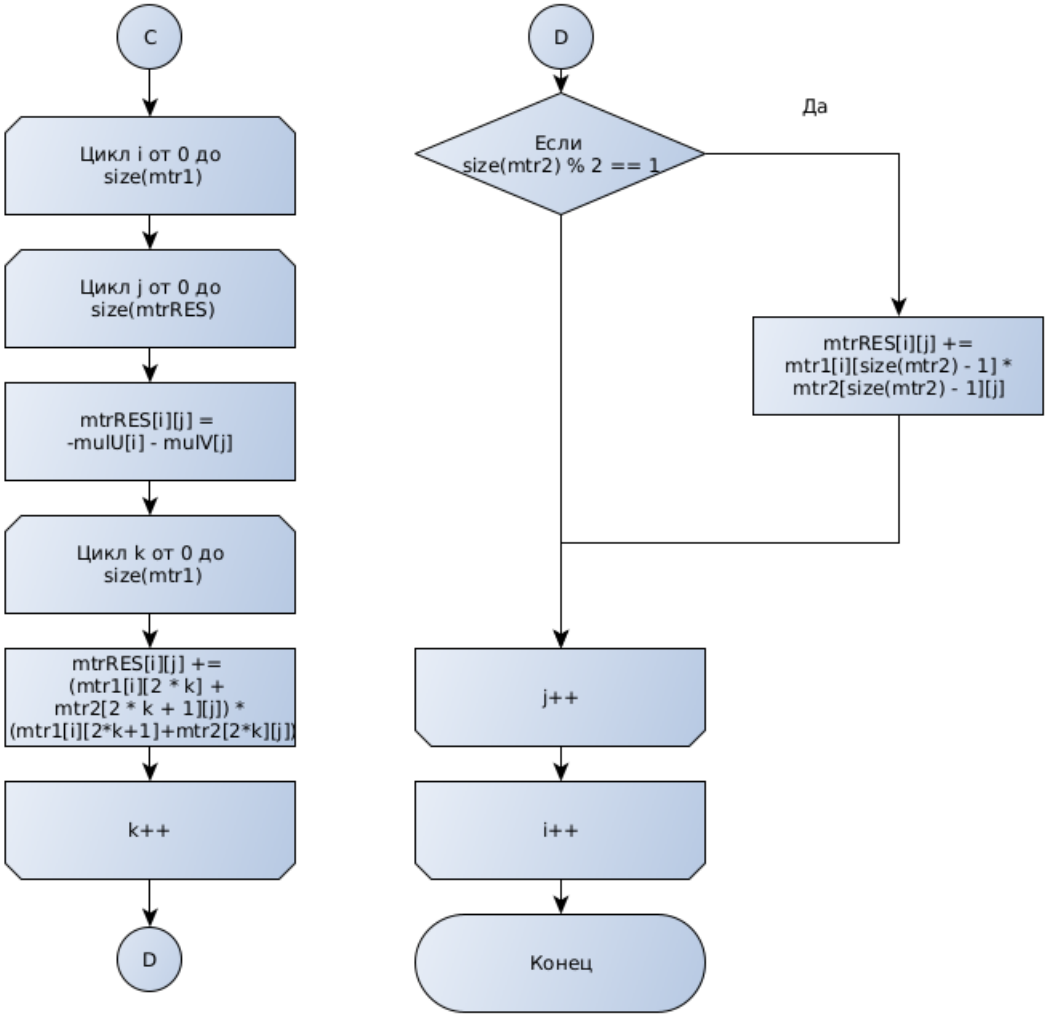

Рис. 2 – Схема алгоритма Винограда часть 2

На рисунке 3 – 4 изображена схема алгоритма Винограда с возможность распараллеливания вычислений.

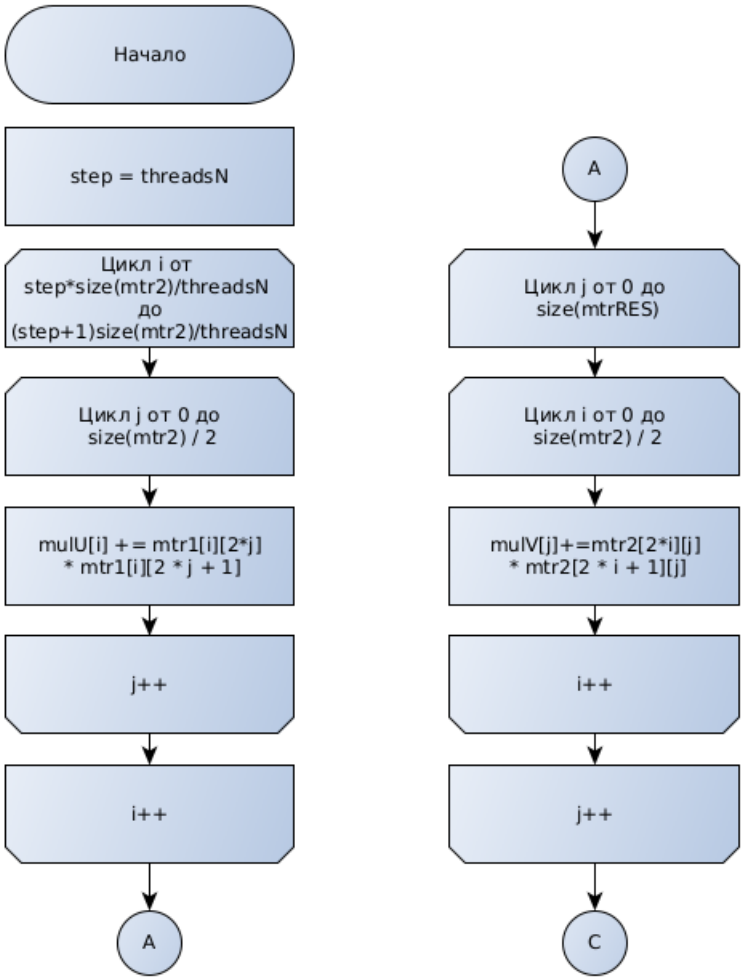

Рис. 3 – Схема алгоритма Винограда с возможностью распараллеливания часть 1

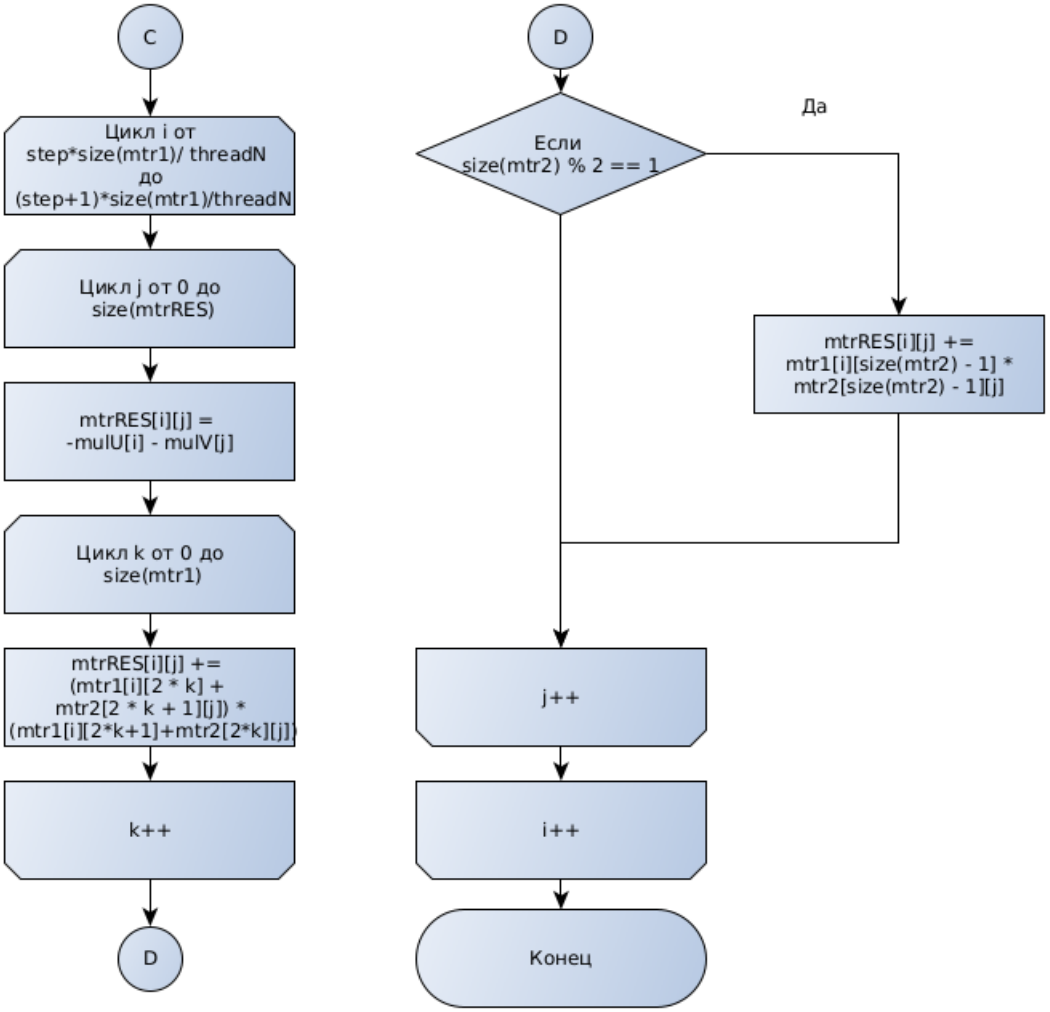

Рис. 4 – Схема алгоритма Винограда с возможностью распараллеливания часть 2

2.2 Выводы

Благодаря возможности вычислять каждый элемент результативной матрицы отдельно друг от друга, удалось разделить вычисления, выдавая каждому потоку диапазон , в котором необходимо считать. После выполнения всех потоков получается правильный результат.

3 Технологическая часть

3.1 Требования к программному обеспечению

Программное обеспечение должно обеспечивать замер процессорного времени выполнения каждого алгоритма. Проводятся замеры для случайно генерируемых квадратных матриц размерности до 1000.

3.2 Средства реализации

В качестве языка программирования был выбран `Kotlin`. Данный язык имеет полную совместимость с Java. Как и Java, C и C++, Kotlin[2] — это статически типизированный язык. Он поддерживает как объектно-ориентированное, так и процедурное программирование. Программа, написанная на `Kotlin`, будет доступна на всех платформах.

3.3 Методы замера времени в программе

3.3.1 Время

Время замерялось с помощью функции `measureTimeMillis`, которая измеряет процессорное время в милисекундах.

Листинг 1 – Функция замера времени.

```
1 inline fun measureTimeMillis(block: () -> Unit): Long
```

Для распараллеливания вычислений была использована функция `thread` [3].

Листинг 2 – Функция создания потока.

```
1 fun thread(
2     start: Boolean = true,
3     isDaemon: Boolean = false,
4     contextClassLoader: ClassLoader? = null,
5     name: String? = null,
6     priority: Int = -1,
7     block: () -> Unit
8 ): Thread
```

Тестирование проводится на процессоре с количеством логических потоков равным 4.

3.4 Листинг кода

Результаты разработки указаны в листингах 3 – 5 .

Листинг 3 – Алгоритм Винограда умножения матриц.

```
fun Multiply(
        firstMatrix: Array<IntArray>,
        secondMatrix: Array<IntArray>,
        resultMatrix: Array<IntArray>,
    ) {
        var firstMatrixSize = firstMatrix.size
        var secondMatrixSize = secondMatrix.size
        var resultMatrixSize = secondMatrix[0].size

        val mulU = Array(firstMatrixSize) { 0 }
        val mulV = Array(resultMatrixSize) { 0 }

        for (i in 0 until firstMatrixSize) {
            for (j in 0 until secondMatrixSize / 2) {
                mulU[i] += firstMatrix[i][2 * j] * firstMatrix[i][2 * j + 1]
            }
        }

        for (j in 0 until resultMatrixSize) {
            for (i in 0 until secondMatrixSize / 2) {
                mulV[j] += secondMatrix[2 * i][j] * secondMatrix[2 * i + 1][j]
            }
        }

        for (i in 0 until firstMatrixSize) {
            for (j in 0 until resultMatrixSize) {
                resultMatrix[i][j] = -mulU[i] - mulV[j]

                for (k in 0 until secondMatrixSize / 2) {
                    resultMatrix[i][j] += (firstMatrix[i][2 * k]
                    + secondMatrix[2 * k + 1][j]) *
                            (firstMatrix[i][2 * k + 1] +
                            secondMatrix[2 * k][j])
                }

                if (secondMatrixSize % 2 == 1) {
                    resultMatrix[i][j] += firstMatrix[i][secondMatrixSize - 1] *
                            secondMatrix[secondMatrixSize - 1][j]
                }
            }
        }
    }
```

Листинг 4 – Функция распараллеливание умножения матриц.

```
fun ThreadMultiplication(
        firstMatrix: Array<IntArray>,
        secondMatrix: Array<IntArray>,
        resultMatrix: Array<IntArray>,
    ) : Long {
        this.firstMatrix = firstMatrix
        this.secondMatrix = secondMatrix
        this.resultMatrix = resultMatrix

        zeroResultMatrix()
        val threadArray = ArrayList<Thread>()

        var time = measureTimeMillis {
            for (i in 0 until numbersOfThread) {
                threadArray.add(thread(start = true){ separetedMultiplication() })
            }

            for (thread in threadArray) {
                thread.join()
            }
        }
        iterationCounter = 0
        return time
    }
```

Листинг 5 – Функция распараллеливание умножения матриц.

```
private fun separetedMultiplication() {
        var step = iterationCounter++;
        var firstMatrixSize = firstMatrix.size
        var secondMatrixSize = secondMatrix.size
        var resultMatrixSize = secondMatrix[0].size

        val mulU = Array(firstMatrixSize) { 0 }
        val mulV = Array(resultMatrixSize) { 0 }

        for (i in step * secondMatrixSize / numbersOfThread
                until (step + 1) * secondMatrixSize / numbersOfThread) {
            for (j in 0 until secondMatrixSize / 2) {
                mulU[i] += firstMatrix[i][2 * j] * firstMatrix[i][2 * j + 1]
            }
        }

        for (j in 0 until resultMatrixSize) {
            for (i in 0 until secondMatrixSize / 2) {
                mulV[j] += secondMatrix[2 * i][j] * secondMatrix[2 * i + 1][j]
            }
        }

        for (i in step * firstMatrixSize / numbersOfThread
                until (step + 1) * firstMatrixSize / numbersOfThread) {
            for (j in 0 until resultMatrixSize) {
                resultMatrix[i][j] = -mulU[i] - mulV[j]
                for (k in 0 until secondMatrixSize / 2) {
                    resultMatrix[i][j] += (firstMatrix[i][2 * k] +
                                           secondMatrix[2 * k + 1][j]) *
                            (firstMatrix[i][2 * k + 1] + secondMatrix[2 * k][j])
                }
                if (secondMatrixSize % 2 == 1) {
                    resultMatrix[i][j] += firstMatrix[i][secondMatrixSize - 1] *
                            secondMatrix[secondMatrixSize - 1][j]
                }
            }
        }
    }
```

Благодаря возможности вычислять каждый элемент результативной матрицы отдельно друг от друга, удалось разделить вычисления. После выполнения всех потоков и соответственно расчетов, получается правильный результат.

3.5 Тестирование

Для тестирования программы были заготовлены следующие тесты в таблице 1.

Таблица 1 – Тесты для алгоритмов

Первая матрца	Вторая матрица	Ожидаемый результат
1 2 3 4	1 2 3 4	7 10 15 22
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	30 36 42 66 81 96 102 126 150
2 0 -1 3	2 0 -1 3	4 0 -5 9

4 Экспериментальная часть

В данном разделе приведены примеры работы программы, постановка эксперимента и сравнительный анализ алгоритмов на основе экспериментальных данных.

4.1 Примеры работ

Пример 1

Матрица A:
1 2 3
4 5 6
Матрица B:
1
2
3
Результирующая матрица:
14
32

Пример 2

Матрица A:
5 2
1 4
Матрица B:
0 3
-6 1
Результирующая матрица:
-12 17
-24 7

Пример 3

Матрица A:
2 7
1 3
Матрица B:
-3 7
1 -2
Результирующая матрица:
1 0
0 1

4.2 Результаты тестирования

Для тестирования были использованы тесты из таблице 1. Результаты продемонстрированы в таблицах 2 и 3.

Таблица 2 – Результаты однопоточного алгоритма

Первая матрца	Вторая матрица	Результат
1 2 3 4	1 2 3 4	7 10 15 22
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	30 36 42 66 81 96 102 126 150
1 2 3 4 5 6	1 2 3	14 32

Таблица 3 – Результаты многопоточного алгоритма

Первая матрца	Вторая матрица	Результат
1 2 3 4	1 2 3 4	7 10 15 22
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	30 36 42 66 81 96 102 126 150
1 2 3 4 5 6	1 2 3	14 32

Все тесты пройдены успешно.

4.3 Замеры времени

В таблице 4 представлены результаты замера времени алгоритмов при четных размерностях, а в таблице 5 при нечетных размерностях матриц. Оба случая прогонялись на квардатных матрицах.

Таблица 4 – Результаты замеры времени на четных размерностях матриц

число потоков \размер	100	200	300	400	500
2	2м.с.	14м.с.	27м.с.	69м.с.	138м.с.
4	1м.с.	10м.с.	24м.с.	59м.с.	117м.с.
8	1м.с.	7м.с.	26м.с.	61м.с.	122м.с.
16	1м.с.	13м.с.	31м.с.	70м.с.	131м.с.
32	16м.с.	15м.с.	36м.с.	75м.с.	226м.с.

Таблица 5 – Результаты замеры времени на нечетных размерностях матриц

число потоков \размер	101	201	301	401	501
2	3м.с.	10м.с.	23м.с.	69м.с.	142м.с.
4	1м.с.	8м.с.	21м.с.	66м.с.	120м.с.
8	1м.с.	7м.с.	22м.с.	62м.с.	130м.с.
16	2м.с.	15м.с.	30м.с.	69м.с.	132м.с.
32	14м.с.	15м.с.	35м.с.	77м.с.	155м.с.

4.4 Выводы

Из таблицы 4 и 5 видно, что число потоков дает въйгрыш во времени до 8 потоков, программа с большим количеством потоков начинает работать медленее. Еще видно, что на матрицах размерностью 100x100, программа работающая с 2 потоками работает в 2 раза медленее, чем программа работающая с 4 потоками, также в 4 раза быстрее, чем программа работающая с 32 потоками.

Заключение

В ходе данной работы было проведено сравнение расчета произведения матриц на нескольких потоках, а именно на 2, 4, 6, 8, 16 и 32 и были сделаны следующие выводы:

- распараллеленные вычисления эффективней в 2 раза;
- исользование числа потоков больше, чем число потоков процессора не дает выйгрыша по времени и может даже работать медленнее.

При выполнения лабораторной работы цель была достигнута и выполнены следующие задачи:

1) изучено распараллеливания вычислений и работа с потоками;
2) реализовано распараллеливание вычислений;
3) выполнено экспериментальное сравнение работы алгоритма на разном количестве потоков.

Список литературы

[1] Умножение матриц: [ЭЛ. РЕСУРС] Режим доступа: Режим доступа: http://www.algolib.narod.ru/Math/Matrix.html. (дата обращения: 03.10.2020).

[2] JetBrains Kotlin [ЭЛ. РЕСУРС] Режим доступа: https://kotlinlang.org/. (дата обращения: 21.10.2020).

[3] Документация по Thread [ЭЛ. РЕСУРС] Режим доступа: https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.concurrent/ (дата обращения: 21.10.2020).