МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ им. В. И. ВЕРНАДСКОГО»
ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
Кафедра компьютерной инженерии и моделирования

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1
«УМНОЖЕНИЕ МАТРИЦ»

Лабораторная работа
по дисциплине
«Параллельные и распределенные вычисления»
студента 4 курса группы ПИ-б-о-182(2)
Змитрович Никита Сергеевич

направления подготовки 09.03.04 «Программная инженерия»

Научный руководитель
старший преподаватель кафедры
компьютерной инженерии и моделирования

_____
(оценка)

_____
(подпись, дата)

Чабанов В.В.

Симферополь, 2021

Цель:

1. Изучить средства управления потоками на платформе JVM
2. Реализовать приложение выполняющее многопоточные вычисления;
3. Сравнить скорость выполнения вычислений при условии использования различного количества потоков

Постановка задачи:

Даны две квадратные матрицы A и B вещественных чисел. Получите матрицу C, которая является произведением матриц A и B. Постройте зависимость ряд зависимостей времени решения задачи от количества использованных потоков.

Выполнение работы

Задание 1.

Данная лабораторная была выполнена при помощи встроенного интерфейса Executor доступного на платформе JVM. Его суть состоит в том, что формируется пул из заранее определённого количества потоков вместо того, чтобы для каждой новой задачи создавать новый поток (хотя никто не запрещает реализовать и такой Executor), а далее на него отправляются задачи (job), который каждый из свободных потоков способен взять. По мере их выполнения поток возвращается обратно в пул свободных потоков и снова доступен для выполнения задач. Данный подход минимизирует штрафы за создание новых потоков.

Для начала составили алгоритм умножение двух матриц на одном потоке. Далее следовало составить алгоритм, который позволит производить параллельные вычисление. Первое, что пришло в голову это создать n^2 задач для параллельных вычислений, однако результаты оказались неутешительными. Один поток в большинстве случаев был быстрее чуть-ли не в 2 раза. Далее было принято решение создать n задач и в таком случае уже результаты начали оправдывать ожидания:

Таблица 1 — Результаты измерений

	1	**2**	**3**	**4**
500	0.326	0.183	0.15	0.13
1000	2.649	1.318	0.902	0.603
2000	47.895	23.02	19.311	11.697

Вывод:

Изучили базовые средства управления потоками на платформе JVM. Реализовали приложение выполняющее параллельное вычисление произведение матриц. Сравнили скорость выполнения вычислений при условий использования различного количества потоков.

Приложение

```
import java.util.concurrent.ExecutorService
import java.util.concurrent.Executors
import java.util.concurrent.TimeUnit
import kotlin.random.Random
import kotlin.system.measureTimeMillis


fun main() {

    println("Введите размерность матрицы: ")
    val n = readLine()!!.toInt()

    require(n > 0) { "n should be > 0" }


    val A = Array(n) { IntArray(n) { Random.nextInt(1, 10) } }
    val B = Array(n) { IntArray(n) { Random.nextInt(1, 10) } }

    println("Доступно ${Runtime.getRuntime().availableProcessors()} потоков")
    println("Введите количество потоков которые следует задействовать в вычислении: ")

    val executors = Executors.newFixedThreadPool(
        readLine()!!.toInt()
    )

    val C = Array(n) { IntArray(n) }

    val calculationTime = measureTimeMillis {
        executors.runBlocking {
            for (row in A.indices) {
                execute {
                    for (column in B[0].indices) {
                        for (i in 0 until n) {
                            C[row][column] += A[row][i] * B[i][column]
                        }
                    }
                }
            }
        }
    }

    print("Вычисления заняли: ")
    print(calculationTime.toDouble() / 1000)
    println(" секунд")

}
```

```
inline fun ExecutorService.runBlocking(block: ExecutorService.() -> Unit) {
    apply(block)
    shutdown().also { awaitTermination(120, TimeUnit.SECONDS) }
}
```