



Міністерство освіти і науки України  
Національний технічний університет України  
“Київський політехнічний інститут імені Ігоря Сікорського”  
Факультет інформатики та обчислювальної техніки  
Кафедра інформаційних систем та технологій

## Лабораторна робота №8

Виконала:

Студентка групи IA-32

Павлюк В.О.

Київ 2025

## **1. Мета роботи**

Метою лабораторної роботи є ознайомлення з методом Монте-Карло для наближеного обчислення числа  $\pi$ , а також вивчення принципів паралельного програмування в мові Java з використанням багатопотоковості для підвищення продуктивності обчислень.

## **2. Теоретичні відомості**

Метод Монте-Карло - це ймовірнісний чисельний метод, який базується на використанні випадкових величин для отримання наблизених результатів математичних обчислень. Точність такого методу залежить від кількості проведених ітерацій.

Для обчислення числа  $\pi$  використовується геометричний підхід. Розглядається одиничний квадрат зі стороною 1 та четверть кола радіуса 1, вписаного в цей квадрат. Площа квадрата дорівнює 1, а площа четверті кола -  $\pi/4$ .

У квадрат випадковим чином генерується велика кількість точок. Дляожної точки обчислюється відстань до початку координат  $(0,0)$ . Якщо точка задовольняє умову  $x^2 + y^2 \leq 1$ ,

то вона знаходиться всередині кола.

Частка точок, що потрапили всередину кола, наближено дорівнює  $\pi/4$ . Відповідно, значення числа  $\pi$  можна обчислити за формулою:

$$\pi \approx 4 \cdot (\text{кількість точок у колі} / \text{загальна кількість точок})$$

Для підвищення швидкодії обчислень метод реалізується паралельно з використанням кількох потоків.

## **3.Хід роботи**

- Було створено Java-програму ParallelMonteCarloPi.java.
- Програма приймає один вхідний аргумент - кількість потоків.
- Загальна кількість ітерацій фіксована та дорівнює 1 000 000 000.
- Загальна кількість ітерацій рівномірно розподіляється між потоками.
- У кожному потоці генеруються випадкові точки в одиничному квадраті.
- Дляожної точки перевіряється, чи потрапляє вона в межі одиничного кола.
- Після завершення роботи всіх потоків результати підсумовуються.
- Обчислюється наближене значення числа  $\pi$  та час виконання програми.

- Результати виводяться у консоль у заданому форматі.

## Лістинг програми

```

import java.text.NumberFormat;
import java.util.Locale;
import java.util.SplittableRandom;

public class ParallelMonteCarloPi {

    private static final long ITERATIONS = 1_000_000_000L;

    private static final long BASE_SEED = 123456789L;

    public static void main(String[] args) throws InterruptedException {
        if (args.length != 1) {
            System.out.println("Usage: java ParallelMonteCarloPi <threads>");
            return;
        }

        final int threads;
        try {
            threads = Integer.parseInt(args[0]);
        } catch (NumberFormatException e) {
            System.out.println("Threads must be an integer.");
            return;
        }
        if (threads <= 0) {
            System.out.println("Threads must be > 0.");
            return;
        }

        final long start = System.nanoTime();

        final long base = ITERATIONS / threads;
        final long rem = ITERATIONS % threads;

        final long[] insideCounts = new long[threads];
        Thread[] workers = new Thread[threads];

        for (int t = 0; t < threads; t++) {
            final int tid = t;
            final long itersForThread = base + (tid < rem ? 1 : 0);

            workers[tid] = new Thread(() -> {
                SplittableRandom rnd = new SplittableRandom(BASE_SEED + tid * 1_000_003L);

                long inside = 0;
                for (long i = 0; i < itersForThread; i++) {
                    double x = rnd.nextDouble(); // [0,1]
                    double y = rnd.nextDouble(); // [0,1]
                    double r2 = x * x + y * y;
                    if (r2 <= 1.0) inside++;
                }
                insideCounts[tid] = inside;
            });
            workers[tid].start();
        }
    }
}

```

```

    }

    for (Thread w : workers) w.join();

    long insideTotal = 0;
    for (long c : insideCounts) insideTotal += c;

    double pi = 4.0 * ((double) insideTotal / (double) ITERATIONS);

    final long end = System.nanoTime();
    double timeMs = (end - start) / 1_000_000.0;

    NumberFormat nf = NumberFormat.getInstance(Locale.US);
    nf.setGroupingUsed(true);

    System.out.printf(Locale.US, "PI is %.5f%n", pi);
    System.out.println("THREADS " + threads);
    System.out.println("ITERATIONS " + nf.format(ITERATIONS));
    System.out.printf(Locale.US, "TIME %.2fms%n", timeMs);
}
}

```

```

PS E:\java-labs\lab8> java ParallelMonteCarloPi 8
PI is 3.14151
THREADS 8
ITERATIONS 1,000,000,000
TIME 508.24ms

```

Рисунок 1 — Результат виконання програми

**Висновок:** У ході виконання лабораторної роботи було реалізовано паралельний алгоритм обчислення числа  $\pi$  методом Монте-Карло. Використання багатопотоковості дозволило значно зменшити час виконання програми при великій кількості ітерацій. Експериментально підтверджено, що зі збільшенням кількості випадкових точок точність наближення зростає, а кількість потоків суттєво впливає на продуктивність програми. Отримані результати підтверджують ефективність паралельних обчислень для задач великого обсягу.