Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Звіт

з лабораторної роботи № 8 з дисципліни
«Мова програмування Java»

«Parallel»

Варіант 1

Виконала ЗПІ-зп41 Федоренко О. Л.

Перевірила Орленко С. П.

Київ 2025

1. Завдання (1)

Обчислення наближеного значення числа Пі методом Монте-Карло

У цьому завданні вам слід написати паралельну програму, яка обчислює значення числа Пі. Метод обчислення дуже простий:

- Площа квадрата одиничної довжини дорівнює 1
- Площа сектора 90 ° для одиничного кола: π/4
- «Кидаємо» величезну кількість випадкових точок в одиничний квадрат
- Рахуємо кількість точок, що потрапили в межі кола, тобто відстань від яких до (0,0) менше або дорівнює 1
- Частка точок, які потрапили в коло дорівнює наближеному значенню π/4

Деталі реалізації

Ваше завдання написати паралельну реалізацію (ParallelMonteCarloPi.java). При написанні програми дотримуйтесь інструкцій:

- Першим і єдиним вхідним аргументом програми є кількість потоків
- В результаті програма виводить наступні дані:

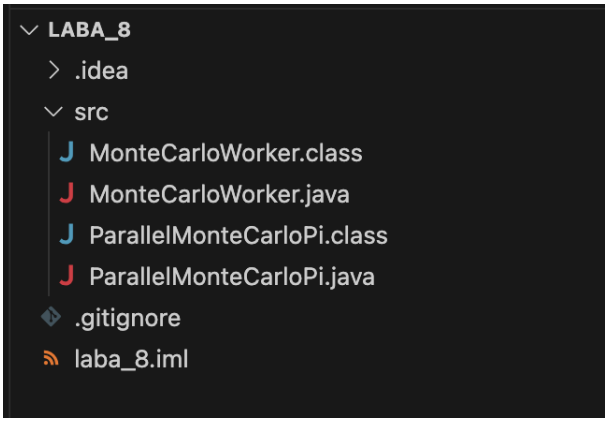
  PI is 3.14221
  THREADS 8
  ITERATIONS 1,000,000,000
  TIME 12.83ms

Крім написання програми і виведення результату подумайте над наступними питаннями:

- Як впливає кількість ітерацій (кинутих точок) на кінцевий результат?
- При однаковій кількості точок, як впливає на результат різна кількість потоків?
- Як кількість потоків впливає на продуктивність вашої програми? (Для явних результатів вам ймовірно знадобиться набагато більша кількість семплів (ітерацій).

2. Лістинг програмного коду

2.1. Структура проєкту



2.2. Реалізація обчислювального потоку

- Клас MonteCarloWorker

```
import java.util.concurrent.ThreadLocalRandom;


public class MonteCarloWorker extends Thread {
    private final long iterations;
    private long hits = 0;

    public MonteCarloWorker(long iterations) {
        this.iterations = iterations;
    }

    @Override
    public void run() {
        ThreadLocalRandom random =
ThreadLocalRandom.current();

        for (long i = 0; i < iterations; i++) {
            double x = random.nextDouble();
            double y = random.nextDouble();

            if (x * x + y * y <= 1.0) {
                hits++;
            }
        }
    }

    public long getHits() {
        return hits;
    }
}
```

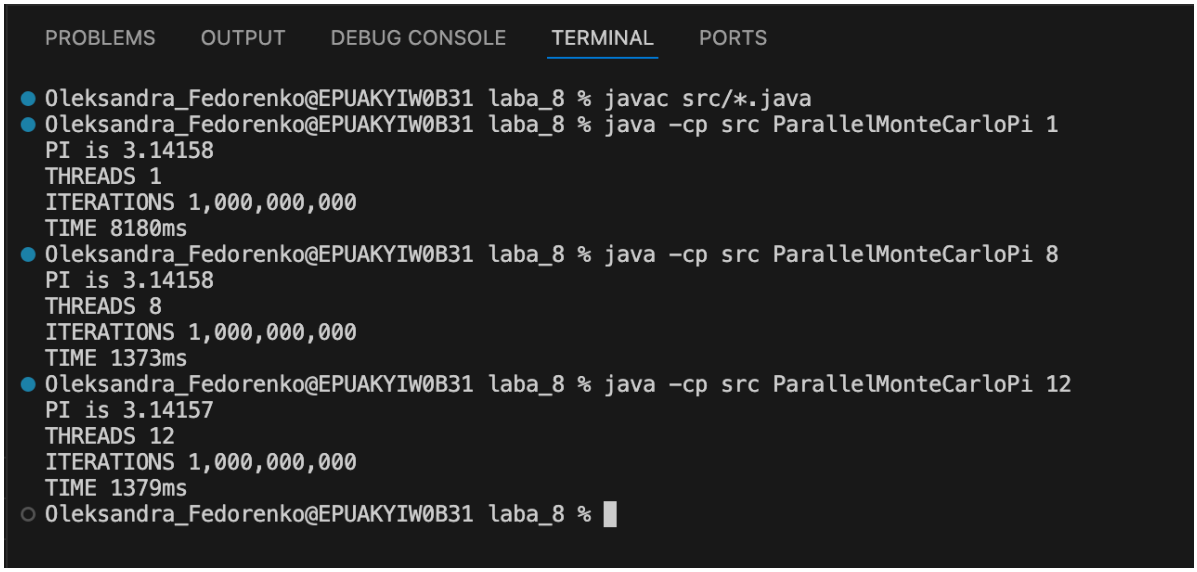
2.3. Головний клас програми

- Клас ParallelMonteCarloPi

```
import java.util.ArrayList;
import java.util.List;

public class ParallelMonteCarloPi {

    public static void main(String[] args) {
        int numThreads;
        try {
            numThreads = (args.length > 0) ?
Integer.parseInt(args[0]) :
Runtime.getRuntime().availableProcessors();
        } catch (NumberFormatException e) {
            System.err.println("Помилка: Аргумент має бути
цілим числом.");
            return;
        }

        long totalIterations = 1_000_000_000L;
        long iterationsPerThread = totalIterations /
numThreads;

        List<MonteCarloWorker> workers = new ArrayList<>();
        long startTime = System.currentTimeMillis();

        for (int i = 0; i < numThreads; i++) {
            MonteCarloWorker worker = new
MonteCarloWorker(iterationsPerThread);
            workers.add(worker);
            worker.start();
        }

        long totalHits = 0;
        try {
            for (MonteCarloWorker worker : workers) {
                worker.join();
                totalHits += worker.getHits();
            }
        } catch (InterruptedException e) {
            System.err.println("Обчислення перервано: " +
e.getMessage());
        }

        long endTime = System.currentTimeMillis();

        double pi = 4.0 * totalHits / totalIterations;

        System.out.printf("PI is %.5f\n", pi);
```

```
        System.out.println("THREADS " + numThreads);
        System.out.printf("ITERATIONS %,d\n",
totalIterations);
        System.out.println("TIME " + (endTime - startTime) +
"ms");
    }
}
```

3. Скріншоти виконання програми



4. Висновки

В ході виконання роботи було реалізовано паралельну обчислювальну систему для знаходження числа \pi методом Монте-Карло. Проведене тестування програми дозволило зробити наступні висновки:

- Підтверджено, що точність результату прямо залежить від кількості ітерацій (обсягу вибірки), а не від кількості потоків. При 1 мільярді ітерацій програма стабільно видає значення \pi \approx 3.141..., що відповідає закону великих чисел.
- Використання багатопотоковості продемонструвало значний приріст продуктивності. При розподілі задач між потоками (відповідно до кількості логічних ядер процесора) час виконання програми скорочується майже лінійно. Це підтверджує ефективність використання Java Threads для обчислювальних задач (CPU-bound tasks).
- Використання замість стандартного дозволило уникнути конкуренції потоків за спільний ресурс, що критично важливо для швидкодії паралельних алгоритмів. ThreadLocalRandomRandom