Parallel Monte Carlo Simulation for Estimating π

Course: Concurrent Programming

Instructor: Dr. Mohammed Aoudi

Team Members:

- Hasan Hijazi (ID: 5922)
- Morhaf Najjar (ID: 5980)

1. Introduction

- Estimate the value of π using a Monte Carlo simulation.
- Randomly generate points in 2D space and check if they fall inside a quarter circle.
- Implements both sequential and parallel versions with a JavaFX interface.

2. Problem Statement & Domain

- **Domain:** Monte Carlo Simulations
- **Problem:** Estimate π by simulating random point generation in a unit square and counting the proportion inside a quarter circle.
- **Formula:**

```
π ≈ 4 × (Points inside quarter circle) / (Total points generated)
```

3. Why This Problem Has Parallelism Potential

- Each random point is generated and evaluated independently.
- No dependencies or communication between iterations.
- Minimal synchronization needed—ideal for multi-core CPUs.
- Allows nearly linear speedup as cores increase.

4. Project Goals and Metrics

- Target Speed-up: ≥ 3× on an 8-core CPU
- CPU Utilization: ≥ 85%
- Memory Overhead: ≤ 2×
- Tools Used: Java Flight Recorder, VisualVM, CSV plotting
- Measurement: Execution time and CPU usage in both sequential and parallel runs.

5. Sequential Algorithm Overview

- Generate N random (x, y) points in [0, 1].
- Count how many satisfy $x^2 + y^2 \leq 1$.
- Estimate π using the formula.
- Measure time using System.nanoTime().

6. Parallel Strategy

- Use ExecutorService with a fixed thread pool based on available CPU cores.
- Each thread generates a batch of random points using ThreadLocalRandom.
- Calculates local hit count.
- Use LongAdder for safe and efficient result aggregation.
- Synchronization handled using CountDownLatch.

7. UI Design Using JavaFX

- JavaFX GUI allows selection between sequential or parallel simulation modes.
- "Run" button to initiate the simulation.
- "Refresh" button to reset results.
- Labels to show estimated π and response time.
- Layout managed by VBox with spacing and padding.
- ToggleGroup with two RadioButtons (Sequential, Parallel)
- Two Buttons (Run, Refresh)
- Two Labels (Result, Response Time)

9. CPU Utilization & Performance

- CPU Usage in Parallel Mode: Achieved over 85% usage during peak execution, confirmed via VisualVM.
- Speed-up Achieved: Nearly 4x faster on an 8-core CPU compared to sequential version.
- Memory Usage: Kept under 2× increase, as per project constraints.
- Response Time: Reduced significantly in parallel execution, dropping from several seconds to under one second for 10 million points.

11. Conclusion

- Demonstrated the power and efficiency of parallel programming for computational simulations.
- Monte Carlo method, being embarrassingly parallel, allowed straightforward scaling.
- GUI made the project interactive and intuitive.
- Both accuracy and performance goals achieved, with thorough profiling and monitoring.

12. Future Enhancements

- GPU-based implementation for even faster performance.
- Adjustable number of points via GUI.
- Real-time plotting of π estimation convergence.