1 2 9 0

UNIVERSIDADE D
COIMBRA

Sistemas Distribuídos 2024/2025
Faculdade de Ciências e Tecnologia
Universidade de Coimbra

Exercises 2: Parallel Programming

1. Serial and Parallel Programming:

In modern distributed systems, parallel programming plays a critical role in improving performance by leveraging multiple processor cores for parallel execution. Understanding how to efficiently use parallelism is essential when designing scalable and high-performance distributed systems.

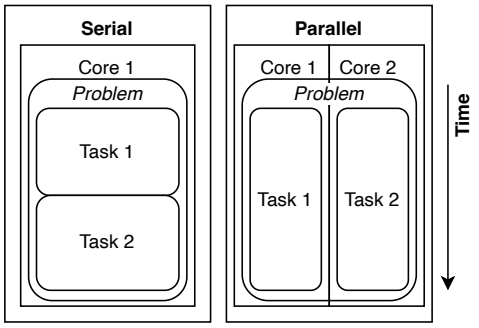

Figure 1: Serial and Parallel Execution

The figure above illustrates the concepts of serial and parallel execution. Serial refers to the ability of a system to handle one task at a time, while parallel refers to the ability to handle multiple tasks simultaneously. Take a look at the programs `Printer.java` and `printer.py`. These example demonstrate how to use serial and parallel processing in Java and Python to print a list of numbers.

1.1. What is the difference between serial and parallel execution?

1.2. Modify the implementation to compute the sum of a list of numbers (use `sum()` function of streams API, and substitute or remove the `forEach()`). Measure the execution time for both serial and parallel versions using appropriate timing functions. What differences do you observe? How does the performance change as the number of elements in the list increases?

1.3. Analyze the difference between serial and parallel processing when each task takes the same execution time. Use a sleep function to simulate uniform task durations. How does parallelization affect execution time in this scenario?

1.4. Implement both a serial version (standard iteration) and a parallel version to calculate the sum of squares of a sequential list with N numbers.

2. Computing π using Monte Carlo Simulation:

The Monte Carlo method is a statistical method that uses random sampling to estimate π. The method simulates a dartboard where darts are thrown at a square board with a circle inscribed. The scenario is a simplified one, with:

- A 1x1 square
- A circle of radius=1

As such, each dart should have a randomly generated (x, y) coordinate where x and y vary between 0..1.

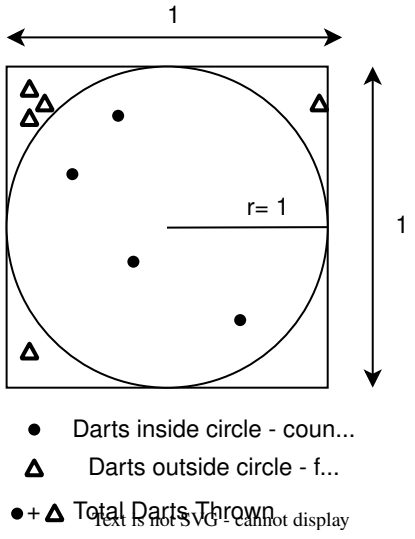

Figure 2: Monte Carlo Pi Estimation

The ratio of darts that land inside the circle to the total number of darts thrown

is used to estimate π. The more darts that are thrown, the more accurate the estimate will be. The formula is as follows:

$$\pi = 4 \times \frac{\textit{Number of thrown darts inside the circle}}{\textit{Number of thrown darts}}$$

2.1. Implement both serial and parallel versions of the Monte Carlo method to estimate π. For this, you need to generate to random values for x, y for each dart, use the streams `filter()` and `count()` methods.

2.2. Measure the execution time for a large numbers of darts (*e.g.*, 10^7, 10^8, 10^9, 10^10). How does the number of darts affect the execution time?

2.3. Observe the Central Processing Unit (CPU) usage while running both implementations. What do you notice?

3. Merge sort of stop words:

Consider the following scenario, where a distributed system is responsible for processing a large collection of keywords. The system must sort the keywords in lexicographical order and remove any stop words from the list. Stop words are common words that are filtered out before or after processing of natural language data. These words do not carry significant meaning and are often removed to improve the performance of text processing algorithms.

3.1. Implement a serial and parallel versions of the merge sort algorithm to sort a list. (Consider using a ForkJoinPool for Java and multiprocessing for Python).

3.2. Try to control the parallelism level of the implementation. Do the performance gains scale linearly with the number of CPUs?