TELUSKO

Parallel Stream

👉 Parallel Stream:

A Parallel Stream in Java is a type of stream that processes elements concurrently using multiple threads, leveraging multi-core processors to improve performance. It is part of the Java Stream API and is built on the Fork/Join framework.

- **Divides the stream into multiple parts** and processes them simultaneously.
- **Uses multiple threads** to execute operations in parallel.
- **Can be created using:**
  1. collection.parallelStream() – Creates a parallel stream from a collection.
  2. stream.parallel() – Converts a sequential stream into a parallel stream.
- **Order of execution is not guaranteed** due to concurrent processing.
- **May improve performance** for large datasets but may have overhead for small collections.
- **Not thread-safe for shared mutable state**, requiring synchronization in such cases.

👉 When to Use Parallel Streams?

- When working with **large datasets** for better performance.
- When **operations are independent** and do not require ordering.
- When **CPU-intensive tasks** need to be distributed across multiple cores.

Note:

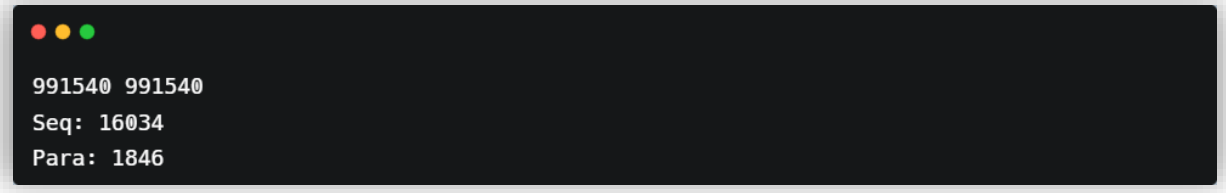
- Avoid parallel stream when processing small datasets.
- Avoid parallel when order of execution is important .
- Avoid parallel when working with shared mutable state, as parallel execution may cause race

TELUSKO

Example:

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

public class Demo {
    public static void main(String[] args) {
        int size = 10_000; // Define the size of the list
        List<Integer> nums = new ArrayList<>();
        Random ran = new Random();

        // Populate the list with random integers between 0 and 99
        for (int i = 1; i <= size; i++) {
            nums.add(ran.nextInt(100));
        }

        // Measure execution time for sequential stream processing
        long startSeq = System.currentTimeMillis();
        int sumSeq = nums.stream()
                .map(i -> {
                    try {
                        Thread.sleep(1); // Simulate processing delay
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    return i * 2;
                })
                .mapToInt(i -> i) // Convert Stream<Integer> to IntStream
                .sum();
        long endSeq = System.currentTimeMillis();

        // Measure execution time for parallel stream processing
        long startPara = System.currentTimeMillis();
        int sumPara = nums.parallelStream()
                .map(i -> {
                    try {
                        Thread.sleep(1); // Simulate processing delay
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    return i * 2;
                })
                .mapToInt(i -> i) // Convert Stream<Integer> to IntStream
                .sum();
        long endPara = System.currentTimeMillis();

        // Print results
        System.out.println("Sequential Sum: " + sumSeq + " | Parallel Sum: " + sumPara);
        System.out.println("Sequential Time: " + (endSeq - startSeq) + "ms");
        System.out.println("Parallel Time: " + (endPara - startPara) + "ms");
    }
}
```

Output:



Note:

The output will be depend on system performance.

TELUSKO