Understanding Java Parallel Streams

What is a Parallel Stream?

In Java, a **parallel stream** is a feature introduced in Java 8 that allows operations on a stream to be executed **concurrently** using multiple threads. Unlike a sequential stream which processes elements one after another, a parallel stream **divides the data into multiple chunks** and processes them **in parallel**, taking advantage of **multi-core CPUs**.

How Does a Parallel Stream Work?

1. **Splitting the Stream Data into Chunks**:
2. Internally, the stream data is split into multiple smaller parts (chunks).
3. This splitting is done using a special interface called `` (short for Splitable Iterator).
4. **Task Submission to Thread Pool**:
5. The chunks are submitted to the **ForkJoinPool**, which is Java's common pool for parallel tasks.
6. ForkJoinPool uses the **work-stealing algorithm** to efficiently balance the load between threads.
7. **Concurrent Execution**:
8. Each chunk is processed independently by separate threads in the pool.
9. Operations like `filter`, `map`, `reduce`, etc., are applied to each chunk in parallel.
10. **Combining the Results**:
11. After all threads finish their work, the results from each chunk are combined back into a single stream (if needed).

Important Component: `Spliterator`

- A `Spliterator` is responsible for breaking the data source into chunks that can be processed in parallel.
- It is used internally by the stream API to support parallel execution.

Parallel Stream Example

```
import java.util.*;

public class ParallelStreamExample {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

        numbers.parallelStream()
               .filter(n -> n % 2 == 0)
               .map(n -> n * 2)
               .forEach(n -> {
                   System.out.println("Thread: " +
Thread.currentThread().getName() + " | Value: " + n);
               });
    }
}
```

Output (Sample)

```
Thread: ForkJoinPool.commonPool-worker-3 | Value: 4
Thread: ForkJoinPool.commonPool-worker-5 | Value: 8
Thread: ForkJoinPool.commonPool-worker-7 | Value: 12
...
```

Note: The actual threads and order may vary in each run.

Key Points to Remember

- Parallel streams use **ForkJoinPool.commonPool** by default.
- The number of threads used is approximately equal to the number of **available CPU cores**.
- Always avoid shared mutable state while using parallel streams.
- Good for CPU-intensive tasks with large datasets.

When to Use Parallel Stream

- When the data is large.
- When the operation is stateless and independent.
- When the processing is CPU-bound (not IO-bound).

Parallel streams can improve performance, but must be used with caution and only when thread-safety and efficiency are ensured.