Leveraging Parallel Streams for Fast Data Processing in Java

INTRODUCING PARALLELISM IN THE JAVA STREAM API

José Paumard

PHD, JAVA CHAMPION, JAVA ROCK STAR

@JosePaumard https://github.com/JosePaumard

```java
double average =
    people.stream()
            .mapToInt(person -> person.getAge())
            .filter(age -> age > 20)
            .average()
            .orElseThrow(); // Java 10+
```

How can we compute this in parallel?

The range version, parallel Stream

```
double average =
   people.stream()
         .mapToInt(person -> person.getAge())
         .filter(age -> age > 20)
         .parallel()
         .average()
         .orElseThrow(); // Java 10+
```

How can we compute this in parallel?

Just call parallel!

How to use parallel Stream

In the right way!

- how data is processed in a parallel Stream
- how the API splits and joins data
- what can affect performances
- detect bottlenecks
- how to choose your source of data

This is a Java course

- basic knowledge of Java

- fair knowledge of the Stream API

- fair knowledge of the Collection API

- how to write lambda expressions

Java version 8+, 11+

Agenda

Writing simple parallel Streams

Measuring Java code performance

Improving the performance in Java

Analyzing the Fork / Join framework

Choosing the right source of data

Agenda

First, let us write a parallel Stream!

How can we measure performance?

Writing a Parallel Stream

```
double average =
   people.stream()
         .mapToInt(person -> person.getAge())
         .filter(age -> age > 20)
         .average()
         .parallel()
         .orElseThrow(); // Java 10+
```

Creating a parallel Stream

```
double average =
   people.parallelStream()
          .mapToInt(person -> person.getAge())
          .filter(age -> age > 20)
          .average()
          // .parallel()
          .orElseThrow(); // Java 10+
```

Creating a parallel Stream

How to measure performances?

Forget about System.*currentTimeInMillis*() or System.*nanoTime*()

The only tool is JMH

`https://openjdk.java.net/projects/code-tools/jmh/`

Let us see an example that works!

We need to heavy computation that will load the CPU

Prime numbers!

```
BigInteger probablePrime(int BIT_LENGTH) {
   return BigInteger.probablePrime(BIT_LENGTH,
                         ThreadLocalRandom.current());
}
```

The value of BIT_LENGTH tunes the size of the prime number

The random values generator provides seeds to generate the prime number

```
List<BigInteger> primes = new ArrayList<>();
for (int i = 0 ; i < N ; i++) {
   primes.add(probablePrime(BIT_LENGTH));
}
```

Let us generate more than one!

The loop version

```
List<BigInteger> primes =
   IntStream(0, N)
       .mapToObj(i -> probablePrime(BIT_LENGTH))
       .collect(Collectors.toList());
```

Let us generate more than one!

The range version, non-parallel Stream

```
List<BigInteger> primes =
   IntStream(0, N)
       .mapToObj(i -> probablePrime(BIT_LENGTH))
       .parallel()
       .collect(Collectors.toList());
```

Let us generate more than one!

The range version, parallel Stream

Measuring Performance with JMH

JMH = Java Microbenchmark Harness

https://github.com/openjdk/jmh

The Open JDK tool to measure performances of application

Running on the JVM

Java, Kotlin, Groovy, Scala, Clojure…

```
@Warmup(iterations = 10, time = 1, timeUnit = TimeUnit.SECONDS)
@Measurement(iterations = 5, time = 1, timeUnit = TimeUnit.SECONDS)
@Fork(value = 3)
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.MILLISECONDS)
@State(Scope.Benchmark)
public class ProbablePrime {
}
```

One way to setup JMH:

1) Annotate a class

```
@State(Scope.Benchmark)
public class ProbablePrime {

    @Param("10", "100")
    private int N;

    @Param("128", "128")
    private int BIT_LENGTH;
}
```

One way to setup JMH:

1) Annotate a class
2) Create parameters

```java
public class ProbablePrime {

   @Benchmark
   public List<BigInteger> rangeParallel() {
      return IntStream(0, N)
            .mapToObj(i -> probablePrime())
            .parallel()
            .collect(Collectors.toList());
   }
}
```

One way to setup JMH:

1) Annotate a class
2) Create parameters
3) Annotate methods

```java
public class ProbablePrime {

   public static void main(String... args) {
      Options options = new OptionBuilder()
                                 .include(ProbablePrime.class)
                                 .build();
      new Runner(options).run();
   }
}
```

One way to setup JMH:

1) Annotate a class
2) Create parameters
3) Annotate methods
4) Run the benchmark

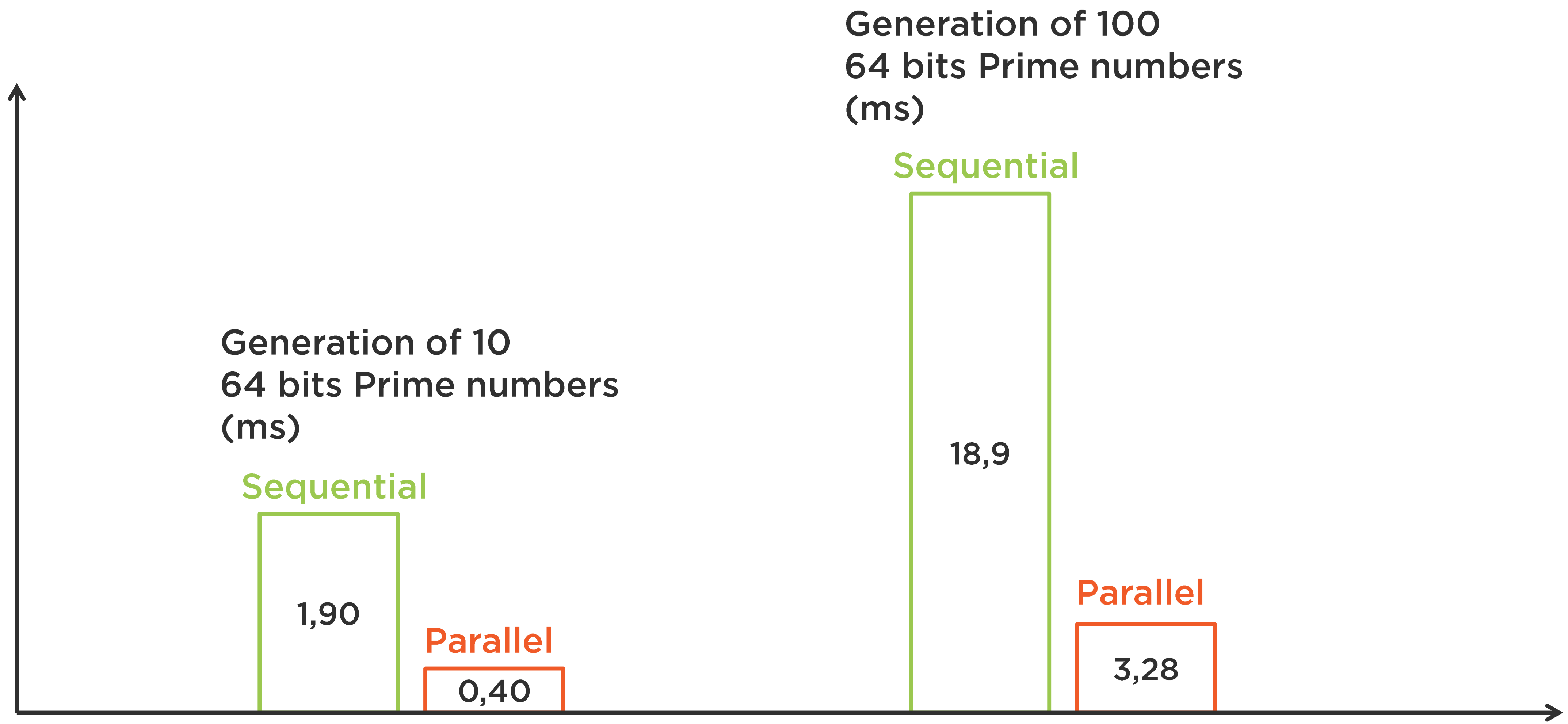
Demo

Let us write some code!

Run our first benchmarks

And see what we can get from parallel Streams!

And result is...



And result is...

```
Benchmark                                   (BIT_LENGTH)  (N)  Mode  Cnt   Score   Error  Units
M02_ProbablePrime.generate_N_primes                   64   10  avgt   15   1,896 ± 0,015  ms/op
M02_ProbablePrime.generate_N_primes                   64  100  avgt   15  18,840 ± 0,066  ms/op
M02_ProbablePrime.generate_N_primes                  128   10  avgt   15   5,668 ± 0,084  ms/op
M02_ProbablePrime.generate_N_primes                  128  100  avgt   15  57,144 ± 0,761  ms/op

M02_ProbablePrime.generate_N_primes_parallel          64   10  avgt   15   0,433 ± 0,017  ms/op
M02_ProbablePrime.generate_N_primes_parallel          64  100  avgt   15   3,281 ± 0,079  ms/op
M02_ProbablePrime.generate_N_primes_parallel         128   10  avgt   15   1,085 ± 0,011  ms/op
M02_ProbablePrime.generate_N_primes_parallel         128  100  avgt   15   8,886 ± 0,149  ms/op
```

Module Wrap Up

What did you learn?

How to create a parallel Stream

How to measure performance with JMH