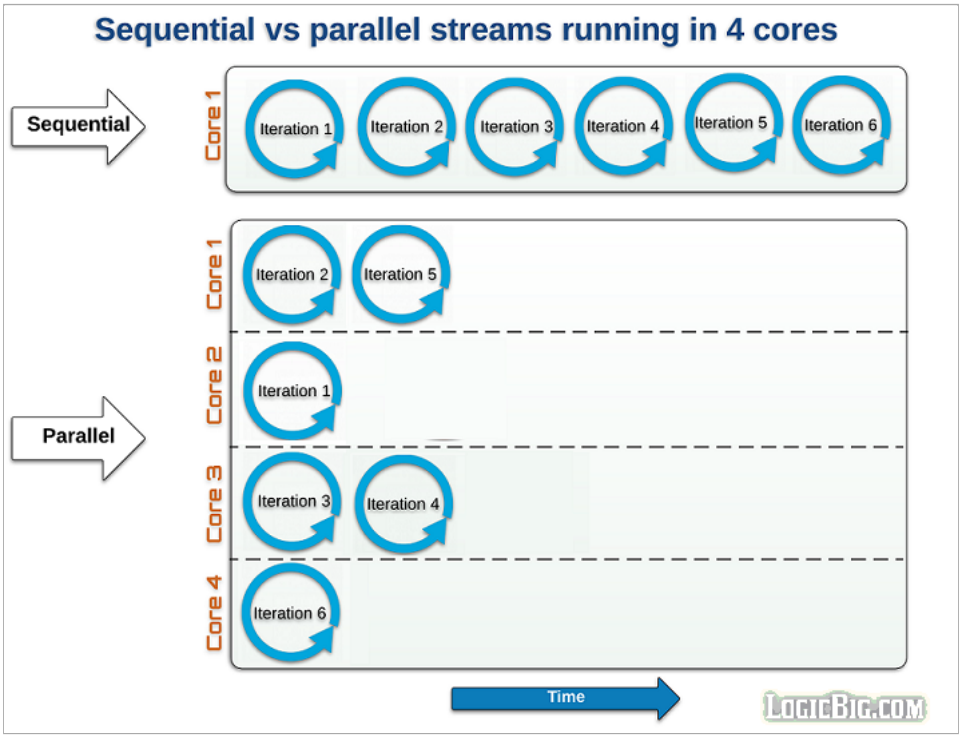
Java 8 Streams - Sequential vs Parallel streams

[Updated: Nov 1, 2018, Created: Sep 19, 2016]

Parallel streams divide the provided task into many and run them in different threads, utilizing multiple cores of the computer. On the other hand sequential streams work just like for-loop using a single core.

The tasks provided to the streams are typically the iterative operations performed on the elements of a collection or array or from other dynamic sources. Parallel execution of streams run multiple iterations simultaneously in different available cores.



In parallel execution, if number of tasks are more than available cores at a given time, the remaining tasks are queued waiting for currently running task to finish.

It is also important to know that iterations are only performed at a terminal operation. The streams are designed to be lazy.

Example

Let's test sequential and parallel behavior with an example.

```
import java.time.LocalTime;
import java.util.Arrays;
import java.util.stream.Stream;

public class SequentialParallelComparison {

    public static void main (String[] args) {
        String[] strings = {"1", "2", "3", "4", "5", "6", "7", "8", "9", "10"};

        System.out.println("-------\nRunning sequential\n-------");
```

```
        run(Arrays.stream(strings).sequential());
        System.out.println("-------\nRunning parallel\n-------");
        run(Arrays.stream(strings).parallel());
    }

    public static void run (Stream<String> stream) {

        stream.forEach(s -> {
            System.out.println(LocalTime.now() + " - value: " + s +
                               " - thread: " + Thread.currentThread().getName());
            try {
                Thread.sleep(200);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });
    }
}
```

In above example we are printing various information, i.e. time, collection element value and thread name. We are doing that in forEach() terminal function. Other than parallel() and sequential(), we are not using any other intermediate operations, but that doesn't matter if we use the same intermediate operations for the both. We are also making each iteration to sleep for 200ms so that we can clearly compare the time taken by sequential and parallel invocations.

Output:

Following is the output, on an 8 logical processors (4 Core) machine.

```
-------
Running sequential
-------
02:29:02.817 - value: 1 - thread: main
02:29:03.022 - value: 2 - thread: main
02:29:03.223 - value: 3 - thread: main
02:29:03.424 - value: 4 - thread: main
02:29:03.624 - value: 5 - thread: main
02:29:03.824 - value: 6 - thread: main
02:29:04.025 - value: 7 - thread: main
02:29:04.225 - value: 8 - thread: main
02:29:04.426 - value: 9 - thread: main
02:29:04.626 - value: 10 - thread: main
-------
Running parallel
-------
02:29:04.830 - value: 7 - thread: main
02:29:04.830 - value: 3 - thread: ForkJoinPool.commonPool-worker-1
02:29:04.830 - value: 8 - thread: ForkJoinPool.commonPool-worker-4
02:29:04.830 - value: 2 - thread: ForkJoinPool.commonPool-worker-3
02:29:04.830 - value: 9 - thread: ForkJoinPool.commonPool-worker-2
02:29:04.830 - value: 5 - thread: ForkJoinPool.commonPool-worker-5
02:29:04.830 - value: 1 - thread: ForkJoinPool.commonPool-worker-6
02:29:04.831 - value: 10 - thread: ForkJoinPool.commonPool-worker-7
02:29:05.030 - value: 4 - thread: ForkJoinPool.commonPool-worker-3
02:29:05.030 - value: 6 - thread: ForkJoinPool.commonPool-worker-2
```

This clearly shows that in sequential stream, each iteration waits for currently running one to finish, whereas, in parallel stream, eight threads are spawn simultaneously, remaining two, wait for others. Also notice the name of threads. In parallel stream, Fork and Join framework is used in the background to create multiple threads. Parallel streams create ForkJoinPool instance via static ForkJoinPool.commonPool() method.

Example project

Dependencies and Technologies Used:

- JDK 1.8
- Maven 3.0.4

Sequential Vs Parallel Streams

```
streams-sequential-vs-parallel
  src
    main
      java
        com
          logicbig
            example
              SequentialParallelCompa
  pom.xml
```

```java
package com.logicbig.example;

import java.time.LocalTime;
import java.util.Arrays;
import java.util.stream.Stream;

public class SequentialParallelComparison {

    public static void main (String[] args) {
        String[] strings = {"1", "2", "3", "4", "5", "6", "7", "8", "9", "10"};

        System.out.println("-------\nRunning sequential\n-------");
        run(Arrays.stream(strings).sequential());
        System.out.println("-------\nRunning parallel\n-------");
        run(Arrays.stream(strings).parallel());
    }

    public static void run (Stream<String> stream) {

        stream.forEach(s -> {
            System.out.println(LocalTime.now() + " - value: " + s +
```

```
                                    " - thread: " + Thread.currentThread().getName());
                try {
                    Thread.sleep(200);
                } catch (InterruptedException e) {
```


Project Structure

```
streams-sequential-vs-parallel
  src
    main
      java
        com
          logicbig
            example
              SequentialParallelComparison.java
  pom.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.logicbig.example</groupId>
    <artifactId>streams-sequential-vs-parallel</artifactId>
    <version>1.0-SNAPSHOT</version>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.5.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                    <encoding>UTF-8</encoding>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

```
package com.logicbig.example;

import java.time.LocalTime;
import java.util.Arrays;
import java.util.stream.Stream;

public class SequentialParallelComparison {

    public static void main (String[] args) {
        String[] strings = {"1", "2", "3", "4", "5", "6", "7", "8", "9", "10"};

        System.out.println("-------\nRunning sequential\n-------");
        run(Arrays.stream(strings).sequential());
        System.out.println("-------\nRunning parallel\n-------");
        run(Arrays.stream(strings).parallel());
    }

    public static void run (Stream<String> stream) {
```

```
        stream.forEach(s -> {
            System.out.println(LocalTime.now() + " - value: " + s +
                               " - thread: " + Thread.currentThread().getName());
            try {
                Thread.sleep(200);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });
    }
}
```


See Also

- Parallel and Sequential streams performance Comparison
- Fork and Join example
- Java 8 Stream operations cheat sheet
- What are Java 8 streams?
- Lazy evaluation
- Short circuiting operations
- Ordering
- Stateful vs Stateless behavioral parameters
- Side Effects
- Reduction
- Mutable Reduction
- Java 8 Streams quick examples

Java 11 Tutorials

- Java 11 Features

Java 10 Tutorials

- Java 10 Features

Java 9 Tutorials

- Java 9 Module System
- Java 9 Misc Features
- Java 9 JShell

Recent Tutorials

- Spring Cloud - Hystrix CircuitBreaker, Thread Local Context Propagation
- Spring Cloud - Circuit Breaker Hystrix Event Listener
- Spring Cloud - Circuit Breaker Hystrix, Changing Default Thread Pool Properties
- Spring Cloud - Circuit Breaker Hystrix, concurrent requests and default thread pool size
- Spring Cloud - Circuit Breaker, Specifying Hystrix configuration in application.properties file
- Spring Cloud - Hystrix Circuit Breaker, Setting Configuration Properties Using @HystrixProperty
- Spring Cloud - Hystrix Circuit Breaker, getting failure exception in fallback method
- Spring Cloud - Circuit Breaker Hystrix Basics
- TypeScript - Standard JavaScript's built-in objects Supp
- JavaScript - Async Processing with JavaScript Promise
- TypeScript - Applying Mixins
- JPA Criteria API - Modifying and Reusing Query Objects
- JPA Criteria API - Delete Operations
- JPA Criteria API - Update Operations
- JPA Criteria API - Case Expressions with CriteriaBuilder.s

Share