# What is Java Parallel Streams?

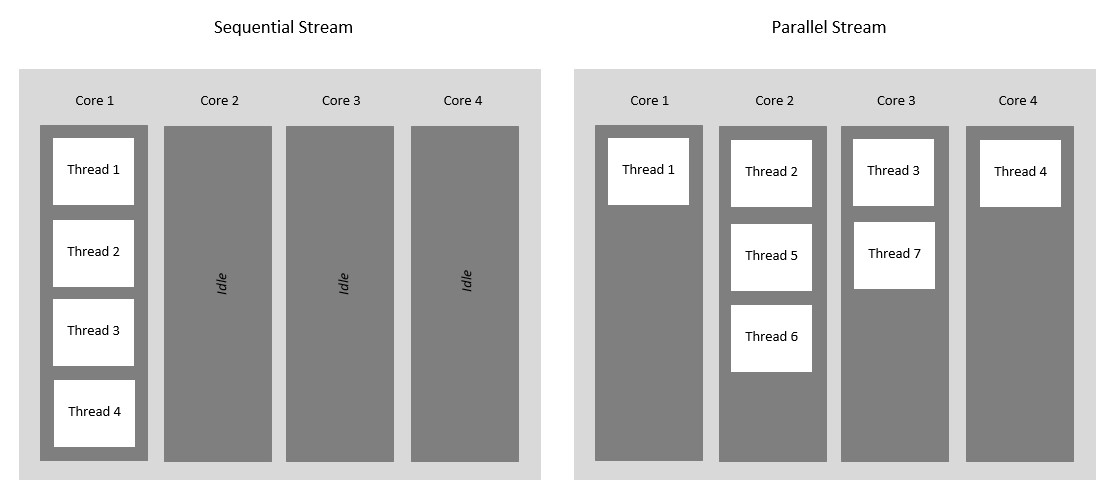
Normally any java code has one stream of processing, where it is executed sequentially. Whereas by using parallel streams, we can divide the code into multiple streams that are executed in parallel on separate cores and the final result is the combination of the individual outcomes

Drawback:

The order of execution, however, is not under our control.

When to use parallel stream?

|  |
| --- |
| Therefore, it is advisable to use parallel streams in cases where no matter what is the order of execution |



### Ways to Create Stream

There are two ways we can create which are listed below and described later as follows:

1. Using parallel() method on a stream
2. Using parallelStream() on a Collection

Example on

**package** main;

**import** java.util.Arrays;

**import** java.util.List;

**public** **class** Main3

{

**public** **static** **void** main(String[] args)

{

List<Integer> l1= Arrays.*asList*(10,20,30,40,50,60,70,80,90,100,101,102,103,104,105,106,107,108,109,110);

l1.stream().forEach((ele)->System.***out***.println(Thread.*currentThread*()+"\t"+ele));

l1.stream().parallel().forEach((ele)->System.***out***.println(Thread.*currentThread*()+"\t"+ele));

l1.parallelStream().forEach((ele)->System.***out***.println(Thread.*currentThread*()+"\t"+ele));

}

}

Output:

Thread[main,5,main] 10

Thread[main,5,main] 20

Thread[main,5,main] 30

Thread[main,5,main] 40

Thread[main,5,main] 50

Thread[main,5,main] 60

Thread[main,5,main] 70

Thread[main,5,main] 80

Thread[main,5,main] 90

Thread[main,5,main] 100

Thread[main,5,main] 101

Thread[main,5,main] 102

Thread[main,5,main] 103

Thread[main,5,main] 104

Thread[main,5,main] 105

Thread[main,5,main] 106

Thread[main,5,main] 107

Thread[main,5,main] 108

Thread[main,5,main] 109

Thread[main,5,main] 110

Thread[main,5,main] 103

Thread[main,5,main] 105

Thread[main,5,main] 104

Thread[main,5,main] 102

Thread[main,5,main] 101

Thread[main,5,main] 108

Thread[main,5,main] 110

Thread[main,5,main] 109

Thread[main,5,main] 107

Thread[main,5,main] 106

Thread[main,5,main] 70

Thread[main,5,main] 60

Thread[ForkJoinPool.commonPool-worker-1,5,main] 80

Thread[main,5,main] 90

Thread[ForkJoinPool.commonPool-worker-1,5,main] 100

Thread[main,5,main] 20

Thread[ForkJoinPool.commonPool-worker-1,5,main] 10

Thread[ForkJoinPool.commonPool-worker-1,5,main] 40

Thread[main,5,main] 50

Thread[ForkJoinPool.commonPool-worker-2,5,main] 30

Thread[main,5,main] 103

Thread[ForkJoinPool.commonPool-worker-3,5,main] 108

Thread[main,5,main] 105

Thread[main,5,main] 104

Thread[ForkJoinPool.commonPool-worker-1,5,main] 102

Thread[ForkJoinPool.commonPool-worker-1,5,main] 101

Thread[ForkJoinPool.commonPool-worker-2,5,main] 30

Thread[ForkJoinPool.commonPool-worker-4,5,main] 70

Thread[ForkJoinPool.commonPool-worker-4,5,main] 60

Thread[ForkJoinPool.commonPool-worker-2,5,main] 50

Thread[ForkJoinPool.commonPool-worker-2,5,main] 40

Thread[ForkJoinPool.commonPool-worker-2,5,main] 80

Thread[ForkJoinPool.commonPool-worker-4,5,main] 90

Thread[ForkJoinPool.commonPool-worker-4,5,main] 107

Thread[ForkJoinPool.commonPool-worker-3,5,main] 110

Thread[ForkJoinPool.commonPool-worker-4,5,main] 106

Thread[ForkJoinPool.commonPool-worker-2,5,main] 100

Thread[ForkJoinPool.commonPool-worker-1,5,main] 10

Thread[main,5,main] 20

Thread[ForkJoinPool.commonPool-worker-3,5,main] 109

This is another example on parallel staream

|  |
| --- |
| **package** main;  **import** java.io.File;  **import** java.io.IOException;  **import** java.nio.file.Files;  **import** java.util.stream.Stream;  **public** **class** ParallelStreamDemo1 {  **public** **static** **void** main(String[] args)**throws** IOException  {    File f=**new** File("info.txt");    **if**(f.exists())  {  System.***out***.println("f.toPath():\t"+f.toPath());  System.***out***.println("f.toPath():\t"+f.getAbsolutePath());  Stream<String> s1=Files.*lines*(f.toPath());  s1.parallel().forEach((line)->System.***out***.println(Thread.*currentThread*()+"\t"+line));    }**else** {  System.***out***.println("File is not existed to print the data...");  }  }  }  Output:  f.toPath(): info.txt  f.toPath(): C:\Users\kalla\eclipse-workspace\App1\info.txt  Thread[ForkJoinPool.commonPool-worker-5,5,main] Vishnu Priya  Thread[main,5,main] Nayanika  Thread[ForkJoinPool.commonPool-worker-3,5,main] Narendra  Thread[ForkJoinPool.commonPool-worker-4,5,main] RamiReddy  Thread[ForkJoinPool.commonPool-worker-7,5,main] NagiReddy  Thread[ForkJoinPool.commonPool-worker-8,5,main] Priya  Thread[ForkJoinPool.commonPool-worker-1,5,main] satwik  Thread[ForkJoinPool.commonPool-worker-2,5,main] Madhu  Thread[ForkJoinPool.commonPool-worker-6,5,main] Vinutna |

Performing intermediate operations and terminal operations on parallel stream

**package** main;

**import** java.io.File;

**import** java.io.IOException;

**import** java.nio.file.Files;

**import** java.util.stream.Stream;

**public** **class** ParallelStreamDemo1 {

**public** **static** **void** main(String[] args)**throws** IOException

{

File f=**new** File("info.txt");

**if**(f.exists())

{

Stream<String> s1=Files.*lines*(f.toPath());

s1.parallel().filter((line)->line.startsWith("N")).forEach((line)->System.***out***.println(Thread.*currentThread*()+"\t"+line));

Stream<String> s2=Files.*lines*(f.toPath());

s2.parallel().filter((line)->line.startsWith("N")).map((line)->"Sri "+line+" Garu").forEach((line)->System.***out***.println(Thread.*currentThread*()+"\t"+line));

}**else** {

System.***out***.println("File is not existed to print the data...");

}

}

}

Output:

Thread[main,5,main] Nayanika

Thread[ForkJoinPool.commonPool-worker-6,5,main] NagiReddy

Thread[ForkJoinPool.commonPool-worker-3,5,main] Narendra

Thread[ForkJoinPool.commonPool-worker-2,5,main] Sri NagiReddy Garu

Thread[main,5,main] Sri Nayanika Garu

Thread[ForkJoinPool.commonPool-worker-1,5,main] Sri Narendra Garu