When to Use (& Not Use) Parallel Streams

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Lesson

- Know when to use parallel streams
Learning Objectives in this Lesson

- Know when to use parallel streams
- & when *not* to use parallel streams
When to Use Java 8
Parallel Streams
When to Use Java 8 Parallel Streams

• A parallel program *always* does more work than a non-parallel program
When to Use Java 8 Parallel Streams

- A parallel program *always* does more work than a non-parallel program, e.g.
- It needs to partition the problem
When to Use Java 8 Parallel Streams

- A parallel program *always* does more work than a non-parallel program, e.g.
  - It needs to partition the problem
  - It needs to perform processing

This step is typically *all* that a sequential program does!
When to Use Java 8 Parallel Streams

• A parallel program *always* does more work than a non-parallel program, e.g.
  • It needs to partition the problem
  • It needs to perform processing
  • It needs to combine the results
When to Use Java 8 Parallel Streams

- A parallel program *always* does more work than a non-parallel program, e.g.
  - It needs to partition the problem
  - It needs to perform processing
  - It needs to combine the results

A sequential program needn’t do the partition & combine steps.

*EXTRA COST*
When to Use Java 8 Parallel Streams

- Java 8 parallel streams are thus useful in some (but not all) conditions

See gee.cs.oswego.edu/dl/html/StreamParallelGuidance.html
When to Use Java 8 Parallel Streams

• Java 8 parallel streams are thus useful in some (but not all) conditions, e.g.
  • When behaviors have certain characteristics
When to Use Java 8 Parallel Streams

- Java 8 parallel streams are thus useful in some (but not all) conditions, e.g.
  - When behaviors have certain characteristics
    - Independent

“Embarrassingly parallel” tasks have little/no dependency or need for communication between tasks or for sharing results between them

See en.wikipedia.org/wiki/Embarrassingly_parallel
When to Use Java 8 Parallel Streams

- Java 8 parallel streams are thus useful in some (but not all) conditions, e.g.
  - When behaviors have certain characteristics
    - Independent
      - e.g., searching for phrases in a list of input strings

```
Input Strings to Search

Search Phrases

map(phrase -> searchForPhrase(…))
filter(not(SearchResults::isEmpty))
collect(toList())
```

When to Use Java 8 Parallel Streams

- Java 8 parallel streams are thus useful in some (but not all) conditions, e.g.
  - When behaviors have certain characteristics
    - Independent
      - e.g., searching for phrases in a list of input strings
When to Use Java 8 Parallel Streams

• Java 8 parallel streams are thus useful in some (but not all) conditions, e.g.
  • When behaviors have certain characteristics
    • Independent
      • e.g., searching for phrases in a list of input strings

Parallel streams can:
  • **search each phrase in parallel**
  • **search each input string in parallel**
  • **search chunks of each input string in parallel**
When to Use Java 8 Parallel Streams

- Java 8 parallel streams are thus useful in some (but not all) conditions, e.g.
  - When behaviors have certain characteristics
  - Independent
    - e.g., searching for phrases in a list of input strings

Parallel streams can:
- search each phrase in parallel
- search each input string in parallel
- search chunks of each input string in parallel
When to Use Java 8 Parallel Streams

- Java 8 parallel streams are thus useful in some (but not all) conditions, e.g.
  - When behaviors have certain characteristics
    - Independent
      - e.g., searching for phrases in a list of input strings

Parallel streams can:
- search each phrase in parallel
- search each input string in parallel
- search chunks of each input string in parallel
When to Use Java 8 Parallel Streams

- Java 8 parallel streams are thus useful in some (but not all) conditions, e.g.
  - When behaviors have certain characteristics
    - Independent
      - e.g., searching for phrases in a list of input strings

**Parallel streams can:**
- search chunks of phrases in parallel
- search chunks of input in parallel
- search chunks of each input string in parallel

SearchWithParallelSpliterator is the most aggressively parallelism strategy!
When to Use Java 8 Parallel Streams

- Java 8 parallel streams are thus useful in some (but not all) conditions, e.g.
  - When behaviors have certain characteristics
    - Independent
  - Computationally expensive
    - e.g., behavior(s) applied to each input element take a “long-time” to run

When to Use Java 8 Parallel Streams

- Java 8 parallel streams are thus useful in some (but not all) conditions, e.g.
  - When behaviors have certain characteristics
    - Independent
    - Computationally expensive
    - Applied to many elements of data sources

When to Use Java 8 Parallel Streams

Java 8 parallel streams are thus useful in some (but not all) conditions, e.g.

- When behaviors have certain characteristics
  - Independent
  - Computationally expensive
- Applied to many elements of data sources
  - Where these sources can be split efficiently/evenly
When to Use Java 8 Parallel Streams

Java 8 parallel streams are thus useful in some (but not all) conditions, e.g.:

- When behaviors have certain characteristics
  - Independent
- Computationally expensive
- Applied to many elements of data sources

The "NQ" model:
- $N$ is the # of data elements to process per thread
- $Q$ quantifies how CPU-intensive the processing is

See on-sw-integration.epischel.de/2016/08/05/parallel-stream-processing-with-java-8-stream-api
When to Use Java 8 Parallel Streams

- Java 8 parallel streams are thus useful in some (but not all) conditions, e.g.
  - When behaviors have certain characteristics
    - Independent
  - Computationally expensive
  - Applied to many elements of data sources

Parallel searching for phrases that match in works of Shakespeare

When to Use Java 8 Parallel Streams

Java 8 parallel streams are thus useful in some (but not all) conditions, e.g.

- When behaviors have certain characteristics
- If there are multiple cores

When to Use Java 8 Parallel Streams

• Under the right conditions Java 8 parallel streams can scale up nicely on multi-core & many-core processors

See www.infoq.com/presentations/parallel-java-se-8
When Not to Use Java 8 Parallel Streams
When Not to Use Java 8 Parallel Streams

- Parallel streams aren’t suitable for certain types of programs

When Not to Use Java 8 Parallel Streams

- Parallel streams aren’t suitable for certain types of programs, e.g.
- The source is expensive to split or splits unevenly

```java
List<CharSequence> arrayAllWords =
    TestDataFactory.getInput(sSHAKESPEARE_WORKS, "\s+");

List<CharSequence> listAllWords =
    new LinkedList<>(arrayAllWords);

arrayAllWords.parallelStream()
    .count();

listAllWords.parallelStream()
    .count();
```

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex14](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex14)
When Not to Use Java 8 Parallel Streams

- Parallel streams aren’t suitable for certain types of programs, e.g.
- The source is expensive to split or splits unevenly

Make an ArrayList that contains all words in the works of Shakespeare

```java
List<CharSequence> arrayAllWords = TestDataFactory.getInput(sSHAKESPEARE_WORKS, "\s+");

List<CharSequence> listAllWords = new LinkedList<>(arrayAllWords);

arrayAllWords.parallelStream()
    .count();

listAllWords.parallelStream()
    .count();
```
When Not to Use Java 8 Parallel Streams

- Parallel streams aren’t suitable for certain types of programs, e.g.
- The source is expensive to split or splits unevenly

```java
List<CharSequence> arrayAllWords = TestDataFactory.getInput(sSHAKESPEARE_WORKS, "\s+");
List<CharSequence> listAllWords = new LinkedList<>(arrayAllWords);
arrayAllWords.parallelStream().count();
listAllWords.parallelStream().count();
```

Make a LinkedList that contains all words in the works of Shakespeare
When Not to Use Java 8 Parallel Streams

- Parallel streams aren’t suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly

```java
List<CharSequence> arrayAllWords =
    TestDataFactory.getInput(sSHAKESPEARE_WORKS, "\s+");

List<CharSequence> listAllWords =
    new LinkedList<>(arrayAllWords);

arrayAllWords.parallelStream()
    .count();

listAllWords.parallelStream()
    .count();
```

The ArrayList parallel stream is much faster than the LinkedList parallel stream

LinkedList performs poorly since it doesn’t try to split evenly/efficiently
When Not to Use Java 8 Parallel Streams

- Parallel streams aren’t suitable for certain types of programs, e.g.
- The source is expensive to split or splits unevenly

```java
class ArrayListSpliterator {
    ...
    ArrayListSpliterator<
    trySplit() {
        int hi = getFence(), lo =
            index, mid = (lo + hi) >>> 1;
        return lo >= mid
            ? null
            : new
                ArrayListSpliterator<
                    (list, lo, index = mid,
                        expectedModCount);
    }
    ...
}
```

See [openjdk/8u40-b25/java/util/ArrayList.java](https://openjdk/8u40-b25/java/util/ArrayList.java)
When Not to Use Java 8 Parallel Streams

- Parallel streams aren’t suitable for certain types of programs, e.g.
- The source is expensive to split or splits unevenly

```java
class LLSpliterator {
    ...
    public Spliterator<E> trySplit() {
        ...
        int n = batch + BATCH_UNIT;
        ...
        Object[] a = new Object[n];
        int j = 0;
        do {
            a[j++] = p.item;
        } while ((p = p.next) != null && j < n);
        ...
        return Spliterators.spliterator(a, 0, j,
                                        Spliterator.ORDERED);
    }

See openjdk/8-b132/java/util/LinkedList.java
```
When Not to Use Java 8 Parallel Streams

- Parallel streams aren’t suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly
  - The startup costs of parallelism overwhelm the amount of data

```java
class ParallelStreamFactorial {
    BigInteger factorial(long n) {
        return LongStream
            .rangeClosed(1, n)
            .parallel()
            .reduce(BigInteger.ONE,
                     BigInteger::multiply);
    }
}
```

```java
class SequentialStreamFactorial {
    BigInteger factorial(long n) {
        return LongStream
            .rangeClosed(1, n)
            .reduce(BigInteger.ONE,
                     BigInteger::multiply);
    }
}
```

See [github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex16](github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex16)
When Not to Use Java 8 Parallel Streams

- Parallel streams aren’t suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly
  - The startup costs of parallelism overwhelm the amount of data

The overhead of creating a parallel stream is greater than the benefits of parallelism for small values of ‘n’

```
class ParallelStreamFactorial {
    BigInteger factorial(long n) {
        return LongStream
            .rangeClosed(1, n)
            .parallel()
            .reduce(BigInteger.ONE,
                     BigInteger::multiply);
    }
}
```

```
class SequentialStreamFactorial {
    BigInteger factorial(long n) {
        return LongStream
            .rangeClosed(1, n)
            .reduce(BigInteger.ONE,
                     BigInteger::multiply);
    }
}
```

When Not to Use Java 8 Parallel Streams

- Parallel streams aren’t suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly
  - The startup costs of parallelism overwhelm the amount of data
  - Combining partial results is costly

List<CharSequence> allWords =
  new LinkedList<>((TestDataFactory.getInput
    (sSHAKESPEARE_DATA_FILE, "\s+")));

... Set<CharSequence> uniqueWords =
  allWords
    .parallelStream()
    ...
    .collect(toCollection
      (TreeSet::new));

See github.com/douglas craig schmidt/LiveLessons/tree/master/Java8/ex14
Parallel streams aren’t suitable for certain types of programs, e.g.

- The source is expensive to split or splits unevenly
- The startup costs of parallelism overwhelm the amount of data
- Combining partial results is costly

```
List<CharSequence> allWords = new LinkedList<>((TestDataFactory.getInput(sSHAKESPEARE_DATA_FILE, "\s+"));
...
Set<CharSequence> uniqueWords = allWords.
.parallelStream()
... .collect(toCollection(TreeSet::new));
```

A linked list of all words in the complete works of Shakespeare
When Not to Use Java 8 Parallel Streams

- Parallel streams aren’t suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly
  - The startup costs of parallelism overwhelm the amount of data
  - Combining partial results is costly

Performance will be poor due to the overhead of combining partial results for a set in a parallel stream

```
List<CharSequence> allWords =
    new LinkedList<>((TestDataFactory.getInput(sSHAKESPEARE_DATA_FILE, "\s+"));

Set<CharSequence> uniqueWords =
    allWords
    .parallelStream()
    .collect(toCollection(TreeSet::new));
```
When Not to Use Java 8 Parallel Streams

- Parallel streams aren’t suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly
  - The startup costs of parallelism overwhelm the amount of data
  - Combining partial results is costly

```java
List<CharSequence> allWords = new LinkedList<>(
    (TestDataFactory.getInput(sSHAKESPEARE_DATA_FILE, "\s+"));

... Set<CharSequence> uniqueWords = allWords
    .parallelStream()
    ... .collect(toCollection(TreeSet::new));
```

The combining cost can be alleviated by the amount of work performed per element (i.e., the “NQ model”)

When Not to Use Java 8 Parallel Streams

• Parallel streams aren’t suitable for certain types of programs, e.g.
  • The source is expensive to split or splits unevenly
  • The startup costs of parallelism overwhelm the amount of data
  • Combining partial results is costly

List<CharSequence> allWords =
        new LinkedList<> (TestDataFactory.getInput (sSHAKESPEARE_DATA_FILE, "\s+"));

... Set<CharSequence> uniqueWords =
          allWords .parallelStream ()
          ... .collect (toSet ());

A concurrent collector can also be used to optimize the reduction phase

See Java8/ex14/src/main/java/utils/ConcurrentHashSetCollector.java
When Not to Use Java 8 Parallel Streams

- Parallel streams aren’t suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly
  - The startup costs of parallelism overwhelm the amount of data
  - Combining partial results is costly
  - A Java 8 feature doesn't enable sufficient exploitable parallelism

List<Double> result = Stream
  .iterate(2, i -> i + 1)
  .parallel()
  .filter(this::isEven)
  .limit(number)
  .map(this::findSQRT)
  .collect(toList());

List<Double> result = LongStream
  .range(2, (number * 2) + 1)
  .parallel()
  .filter(this::isEven)
  .mapToObj(this::findSQRT)
  .collect(toList());

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex15](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex15)
When Not to Use Java 8 Parallel Streams

- Parallel streams aren’t suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly
  - The startup costs of parallelism overwhelm the amount of data
  - Combining partial results is costly
  - A Java 8 feature doesn’t enable sufficient exploitable parallelism

See `github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex15`

```java
List<Double> result = Stream.iterate(2, i -> i + 1)
    .parallel()
    .filter(this::isEven)
    .limit(number)
    .map(this::findSQRT)
    .collect(toList());
```

```java
List<Double> result = LongStream.range(2, (number * 2) + 1)
    .parallel()
    .filter(this::isEven)
    .mapToObj(this::findSQRT)
    .collect(toList());
```

`Stream.iterate()` & `limit()` split & parallelize poorly...
When Not to Use Java 8 Parallel Streams

- Parallel streams aren’t suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly
  - The startup costs of parallelism overwhelm the amount of data
  - Combining partial results is costly
  - A Java 8 feature doesn't enable sufficient exploitable parallelism

List<Double> result = Stream
 .iterate(2, i -> i + 1)
 .parallel()
 .filter(this::isEven)
 .limit(number)
 .map(this::findSQRT)
 .collect(toList());

List<Double> result = LongStream
 .range(2, (number * 2) + 1)
 .parallel()
 .filter(this::isEven)
 .mapToObj(this::findSQRT)
 .collect(toList());

\textit{LongStream.range()} splits nicely & thus runs efficiently in parallel

See \texttt{github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex15}
When Not to Use Java 8 Parallel Streams

- Parallel streams aren’t suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly
  - The startup costs of parallelism overwhelm the amount of data
  - Combining partial results is costly
  - A Java 8 feature doesn't enable sufficient exploitable parallelism
  - There aren’t many/any cores

*Older computing devices just have a single core, which limits available parallelism*
When Not to Use Java 8 Parallel Streams

- Parallel streams aren’t suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly
  - The startup costs of parallelism overwhelm the amount of data
  - Combining partial results is costly
  - A Java 8 feature doesn't enable sufficient exploitable parallelism
  - There aren’t many/any cores
  - No built-in means to shutdown processing of a parallel stream
End of When to Use (& Not Use) Parallel Streams