Java 8 Parallel SearchStreamGang

Example (Part 3)

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 Part of the Lesson

- Know how Java 8 parallel streams are applied in the SearchStreamGang
- Understand the pros & cons of the SearchWithParallelStreams class
- Recognize how a parallel spliterator can improve parallel stream performance
- Understand the pros & cons of the SearchWithParallelSpliterator class
- Know when to use parallel streams
Learning Objectives in this Part of the Lesson

- Know how Java 8 parallel streams are applied in the SearchStreamGang
- Understand the pros & cons of the SearchWithParallelStreams class
- Recognize how a parallel spliterator can improve parallel stream performance
- Understand the pros & cons of the SearchWithParallelSpliterator class
- 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

![Diagram showing parallel processing with joins](image)
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 properties
    - Independent

See en.wikipedia.org/wiki/Embarrassingly_parallel

“Embarrassingly parallel” tasks have little/no dependency or need for communication between tasks or for sharing results between them
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 properties
    - Independent
  - e.g., searching for phrases in a list of input strings

```java
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 properties
    - 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 properties
    • 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
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 properties
    • 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
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 properties
    - 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
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 properties
- 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 concurrent 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 properties
    - Independent
  - Computationally expensive
    - e.g., behaviors 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 properties
    - 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 properties
    - Independent
  - Computationally expensive
  - Applied to many elements of data sources

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 properties
  - Independent
- Computationally expensive
- Applied to many elements of data sources

See `SearchStreamGang/src/main/java/livelessons/utils/PhraseMatchSpliterator.java`

- e.g., `PhraseMatchSpliterator` splits input strings into chunks that it searches for regex matches in parallel

- `trySplit()`: source method
- `collect()`: sink method
- `DataSource`:
  - `trySplit()`: try to split a stream into two
  - `collect()`: collect results
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 properties
  - If there are multiple cores

See blog.oio.de/2016/01/22/parallel-stream-processing-in-java-8-performance-of-sequential-vs-parallel-stream-processing
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](http://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

```
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

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 splits poorly since finding the midpoint requires traversing ½ the list
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<E>
    trySplit() {
        int hi = getFence(), lo = index, mid = (lo + hi) >>> 1;
        return lo >= mid
            ? null
            : new
                ArrayListSpliterator<E>(list, lo, index = mid,
                expectedModCount);
    }

    ...
}
```

The ArrayList splitter runs in \( O(1) \) constant time

See 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);
    }
}
```

The LinkedList spliterator runs in $O(n)$ linear time

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);
    }
}
```
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);
    }
    ...
}

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

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

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));
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+"));

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

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

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex14](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
  - 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
  - A Java 8 feature doesn't enable sufficient exploitable parallelism

```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());
```

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

List<Double> result = Stream
  .iterate(2, i -&gt; 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
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());

LongStream.range() splits nicely & thus runs efficiently in parallel

See 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*
End of Java 8 Parallel SearchStreamGang Example (Part 3)