Java 8 SearchWithParallelSpliterator

Example (Part 1)

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

• Recognize how a parallel spliterator can improve parallel stream performance

```java
SearchResults searchForPhrase(..., boolean parallel) {
    return new SearchResults(..., StreamSupport.stream
        (new PhraseMatchSpliterator(...), parallel)
        .collect(toList()));
}
```
Learning Objectives in this Part of the Lesson

- Recognize how a parallel spliterator can improve parallel stream performance
- This solution fixes a “con” (limited performance) covered earlier

See "Java 8 SearchWithParallelStreams Example"
Overview of SearchWith ParallelSpliterator
Overview of SearchWithParallelSpliterator

- SearchWithParallelSpliterator is yet another implementation strategy in the SearchStreamGang program

See SearchStreamGang/src/main/java/livelessons/streamgangs/SearchWithParallelSpliterator.java
Overview of SearchWithParallelSpliterator

- SearchWithParallelSpliterator uses parallel streams in three ways

<<Java Class>>

SearchWithParallelSpliterator

- processStream(): List<List<SearchResults>>
- processInput(CharSequence): List<SearchResults>
Overview of SearchWithParallelSpliterator

- SearchWithParallelSpliterator uses parallel streams in three ways
- Search chunks of input in parallel

```
Input Strings

Input Strings₁₁
Input Strings₁₂
  Process sequentially

Input Strings₂₁
Input Strings₂₂
  Process sequentially
```

A pool of worker threads
Overview of SearchWithParallelSpliterator

- SearchWithParallelSpliterator uses parallel streams in three ways
  - Search chunks of input in parallel
  - Search chunks of phrases in parallel

![Diagram showing process of searching phrases in parallel with a pool of worker threads.](image-url)
Overview of SearchWithParallelSpliterator

- SearchWithParallelSpliterator uses parallel streams in three ways
  - Search chunks of input in parallel
  - Search chunks of phrases in parallel
  - Search chunks of *each* input string in parallel

![Diagram of parallel processing](image-url)
Overview of SearchWithParallelSpliterator

• SearchWithParallelSpliterator uses parallel streams in three ways
  • Search chunks of input in parallel
  • Search chunks of phrases in parallel
  • Search chunks of each input string in parallel

SearchWithParallelSpliterator is thus the most aggressively parallelism strategy!
Overview of SearchWithParallelSpliterator

The relative contribution of each parallel streams model is shown here:

Time for 38 strings = 703 ms (parallelSpliterator|parallelPhrases|parallelInput)
Time for 38 strings = 738 ms (parallelSpliterator|parallelPhrases|sequentialInput)
Time for 38 strings = 761 ms (sequentialSpliterator|parallelPhrases|parallelInput)
Time for 38 strings = 780 ms (sequentialSpliterator|parallelPhrases|sequentialInput)
Time for 38 strings = 1008 ms (parallelSpliterator|sequentialPhrases|parallelInput)
Time for 38 strings = 1617 ms (parallelSpliterator|sequentialPhrases|sequentialInput)
Time for 38 strings = 1986 ms (sequentialSpliterator|sequentialPhrases|parallelInput)
Time for 38 strings = 3000 ms (sequentialSpliterator|sequentialPhrases|sequentialInput)

See github.com/douglasraugeschmidt/LiveLessons/tree/master/SearchStreamSpliterator
Overview of SearchWithParallelSpliterator

- Longer input strings leverage the parallel spliterator even better:
  - Time for 2 strings = 700 ms (parallelSpliterator|parallelPhrases|parallelInput)
  - Time for 2 strings = 706 ms (sequentialSpliterator|parallelPhrases|parallelInput)
  - Time for 2 strings = 726 ms (parallelSpliterator|sequentialPhrases|parallelInput)
  - Time for 2 strings = 739 ms (sequentialSpliterator|sequentialPhrases|parallelInput)
  - Time for 2 strings = 749 ms (sequentialSpliterator|parallelPhrases|sequentialInput)
  - Time for 2 strings = 759 ms (parallelSpliterator|parallelPhrases|sequentialInput)
  - Time for 2 strings = 1760 ms (parallelSpliterator|sequentialPhrases|sequentialInput)
  - Time for 2 strings = 2870 ms (sequentialSpliterator|sequentialPhrases|sequentialInput)

Longer strings may provide better opportunity to leverage benefits of parallelism
Overview of SearchWithParallelSpliterator

- SearchWithParallelSpliterator processInput() has just one minuscule change

```java
List<SearchResults> processInput(CharSequence inputSeq) {
    String title = getTitle(inputString);
    CharSequence input = inputSeq.subSequence(...);

    List<SearchResults> results = mPhrasesToFind
        .parallelStream()
        .map(phase ->
            searchForPhrase(phase, input, title, true))
        .filter(not(SearchResults::isEmpty))
        .collect(toList());

    return results;
}
```

The value of "true" triggers the use of a parallel search for a phrase in an input string
Overview of SearchWithParallelSpliterator

- searchForPhrase() uses a parallel spliterator to break the input into "chunks" that are processed in parallel

SearchResults searchForPhrase(String phrase, CharSequence input, String title, boolean parallel) {
    return new SearchResults(..., ..., phrase, title, StreamSupport
        .stream(new PhraseMatchMatchSpliterator(input, phrase),
               parallel)
          .collect(toList()));
}
Overview of SearchWithParallelSpliterator

- `searchForPhrase()` uses a parallel spliterator to break the input into "chunks" that are processed in parallel

```java
SearchResults searchForPhrase(String phrase, CharSequence input, String title, boolean parallel) {
    return new SearchResults(..., ..., phrase, title, 
        StreamSupport.stream(new PhraseMatchSpliterator(input, phrase), 
        parallel).collect(toList()));
}
```

See [docs.oracle.com/javase/8/docs/api/java/util/stream/StreamSupport.html#stream](https://docs.oracle.com/javase/8/docs/api/java/util/stream/StreamSupport.html#stream)
Overview of SearchWithParallelSpliterator

- searchForPhrase() uses a parallel spliterator to break the input into “chunks” that are processed in parallel

```java
SearchResults searchForPhrase(String phrase, CharSequence input, String title, boolean parallel) {
    return new SearchResults(..., ..., phrase, title, StreamSupport
        .stream(new PhraseMatchSpliterator(input, phrase),
            parallel)
        .collect(toList()));
}
```

The value of “parallel” is true when searchForPhrase() is called in the SearchWithParallelSpliterator program

See [docs.oracle.com/javase/8/docs/api/java/util/stream/StreamSupport.html#stream](http://docs.oracle.com/javase/8/docs/api/java/util/stream/StreamSupport.html#stream)
Using Parallel Spliterator in SearchStreamGang
Using a Parallel Spliterator in SearchStreamGang

- SearchStreamGang uses PhraseMatchSpliterator that works for both sequential & parallel streams

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

Input Strings to Search

Search Phrases

Using a Parallel Spliterator in SearchStreamGang

- SearchStreamGang uses PhraseMatchSpliterator that works for both sequential & parallel streams
- We focused on the sequential portions earlier

See "Java 8 Sequential SearchStreamGang Example (Part 2)"
Using a Parallel Spliterator in SearchStreamGang

- SearchStreamGang uses PhraseMatchSpliterator that works for both sequential & parallel streams
- We focused on the sequential portions earlier
- We’ll cover the parallel portions now

The goal is to further optimize the performance of the parallel streams solution
Using a Parallel Spliterator in SearchStreamGang

- Here's the input/output of PhraseMatchSpliterator for SearchWithParallelSpliterator

```
List<String>
Stream<String>
Stream<SearchResults>
Stream<SearchResults>
List<SearchResults>
```

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

© Here's the input/output of PhraseMatchSpliterator for SearchWithParallelSpliterator

Using a Parallel Spliterator in SearchStreamGang

```
List<String>
Stream<String>
Stream<SearchResults>
Stream<SearchResults>
List<SearchResults>
```

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

© Here's the input/output of PhraseMatchSpliterator for SearchWithParallelSpliterator

Using a Parallel Spliterator in SearchStreamGang

```
List<String>
Stream<String>
Stream<SearchResults>
Stream<SearchResults>
List<SearchResults>
```

```
parallelStream()
map(phrase -> searchForPhrase(...))
filter(not(SearchResults::isEmpty))
collect(toList())
```
Using a Parallel Spliterator in SearchStreamGang

- Here's the input/output of PhraseMatchSpliterator for SearchWithParallelSpliterator

```
"Brevity is the soul of wit" at [54739]
```

```
My liege, and madam, to expostulate
What majesty should be, what duty is,
Why day is day, night is night, and time is time.
Were nothing but to waste night, day, and time.
Therefore, since brevity is the soul of wit,
And tediousness the limbs and outward flourishes,
I will be brief. Your noble son is mad.
Mad call I it; for, to define true madness,
What is't but to be nothing else but mad?
But let that go....
```

This splitter splits the input into multiple chunks & searches them in parallel
Here's the input/output of PhraseMatchSpliterator for SearchWithParallelSpliterator

“... My liege, and madam, to expostulate What majesty should be, what duty is, Why day is day, night is night, and time is time. Were nothing but to waste night, day, and time. Therefore, since *brevity is the soul of wit*,”

“And tediousness the limbs and outward flourishes, I will be brief. Your noble son is mad. Mad call I it; for, to define true madness, What is't but to be nothing else but mad? But let that go....”

When the split occurs efficiently/evenly the speedups can be substantial!
Using a Parallel Spliterator in SearchStreamGang

- Here’s the input/output of PhraseMatchSpliterator for SearchWithParallelSpliterator

  “... My liege, and madam, to expostulate What majesty should be, what duty is, Why day is day, night is night, and time is time. Were nothing but to waste night, day, and time. Therefore, since brevity is the soul of”

  “wit, And tediousness the limbs and outward flourishes, I will be brief. Your noble son is mad. Mad call I it; for, to define true madness, What is't but to be nothing else but mad? But let that go....”

However, the spliterator must be careful not to split input *across* phrases...
Using a Parallel Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```java
class PhraseMatchSpliterator implements Spliterator<Result> {  
    private CharSequence mInput;

    private final String mPhrase;
    private final Pattern mPattern;
    private Matcher mPhraseMatcher;

    private final int mMinSplitSize;

    private int mOffset = 0;
    ...
```

**Spliterator** is an interface that defines eight methods, including `tryAdvance()` & `trySplit()`

See [SearchStreamGang/src/main/java/livelessons/utils/PhraseMatchSpliterator.java](SearchStreamGang/src/main/java/livelessons/utils/PhraseMatchSpliterator.java)
Using a Parallel Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string.

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    private CharSequence mInput;
    private final String mPhrase;
    private final Pattern mPattern;
    private Matcher mPhraseMatcher;
    private final int mMinSplitSize;
    private int mOffset = 0;
    ...

    These fields implement PhraseMatchSpliterator for both sequential & parallel use-cases.

Some fields are updated in the trySplit() method, which is why they aren’t final.
```
Using a Parallel Spliterator in SearchStreamGang

• PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string.

class PhraseMatchSpliterator implements Spliterator<Result> {
  private CharSequence mInput;

  private final String mPhrase;

  private final Pattern mPattern;

  private Matcher mPhraseMatcher;

  private final int mMinSplitSize;

  private int mOffset = 0;

  ...

  Contains a single work of Shakespeare
Using a Parallel Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string.

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    private CharSequence mInput;

    private final String mPhrase;
    private final Pattern mPattern;
    private Matcher mPhraseMatcher;
    private final int mMinSplitSize;

    private int mOffset = 0;
    ...
```

Contains the phrase to search for in the work
• PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string.

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    private CharSequence mInput;
    private final String mPhrase;
    private final Pattern mPattern;
    private Matcher mPhraseMatcher;
    private final int mMinSplitSize;
    private int mOffset = 0;
    ...
}
```

Contains the regular expression representation of the phrase

See docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html
Using a Parallel Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string.

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    private CharSequence mInput;
    private final String mPhrase;
    private final Pattern mPattern;
    private Matcher mPhraseMatcher;
    private final int mMinSplitSize;
    private int mOffset = 0;
    ...
}
```

Contains a matcher that searches for the phrase in the input.

See [docs.oracle.com/javase/8/docs/api/java/util/regex/Matcher.html](http://docs.oracle.com/javase/8/docs/api/java/util/regex/Matcher.html)
PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    private CharSequence mInput;
    private final String mPhrase;
    private final Pattern mPattern;
    private Matcher mPhraseMatcher;
    private final int mMinSplitSize;
    private int mOffset = 0;
    ...
```

Using a Parallel Spliterator in SearchStreamGang

- Dictates the minimum size to perform a split
Using a Parallel Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string.

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    private CharSequence mInput;

    private final String mPhrase;

    private final Pattern mPattern;

    private Matcher mPhraseMatcher;

    private final int mMinSplitSize;

    private int mOffset = 0;

    ...
}
```

The offset needed to return the appropriate index into the original input string.

This value is reset by each spliterator to account for different chunks.
• PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string
class PhraseMatchSpliterator implements Spliterator<Result> {
  ...
  PhraseMatchSpliterator(CharSequence input, String phrase) {
    String regexPhrase = "\\b" + phrase.trim().replaceAll("\\s+", "\\\\b\\\\\\s+\\\\\\b") + "\\b"; ...

    mPattern = Pattern.compile(regexPhrase,
                                Pattern.CASE_INSENSITIVE | Pattern.DOTALL);
    mPhraseMatcher = mPattern.matcher(input);
    mInput = input; mPhrase = phrase;
    mMinSplitSize = input.length() / 2;
  }
  ...

A regex is compiled into a pattern that matches a phrase across lines

See docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html
Using a Parallel Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```java
class PhraseMatchSpliterator implements Spliterator<SearchResults> {
    ...
    PhraseMatchSpliterator( CharSequence input, String phrase ) {
        String regexPhrase = "\\b" + phrase.trim().replaceAll("\\s+", "\\b\\s+\\b") + "\\b";

        mPattern = Pattern.compile( regexPhrase, Pattern.CASE_INSENSITIVE | Pattern.DOTALL );
        mPhraseMatcher = mPattern.matcher( input );
        mInput = input; mPhrase = phrase;
        mMinSplitSize = input.length() / 2;
    }
    ...
}
```

See docs.oracle.com/javase/8/docs/api/java/util/regex/Matcher.html

A matcher is created to search the input for the regex pattern
Using a Parallel Spliterator in SearchStreamGang

• PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string.

class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    PhraseMatchSpliterator(CharSequence input, String phrase) {
        String regexPhrase = "\\b" + phrase.trim().replaceAll("\\s+", "\\\\b\\\\s+\\\\\\b") + "\\b"; ...
        mPattern = Pattern.compile(regexPhrase,
            Pattern.CASE_INSENSITIVE | Pattern.DOTALL);
        mPhraseMatcher = mPattern.matcher(input);
        mInput = input; mPhrase = phrase;
        mMinSplitSize = input.length() / 2;
    }
    ...

Define the min split size
PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string.

class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    boolean tryAdvance(Consumer<? super Result> action) {
        if (!mPhraseMatcher.find())
            return false;
        else {
            action.accept(new Result(
                mOffset + mPhraseMatcher.start()));
            return true;
        }
    }
    ...
}

Called by the Java 8 streams framework to attempt to advance the spliterator by one word match.

See docs.oracle.com/javase/8/docs/api/java/util/Spliterator.html#tryAdvance
Using a Parallel Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string.

class PhraseMatchSpliterator implements Spliterator<Result> {
    ...

    boolean tryAdvance(Consumer<? super Result> action) {
        if (!mPhraseMatcher.find())
            return false;
        else {
            action.accept(new Result
                (mOffset + mPhraseMatcher.start()));
            return true;
        }
    }

    ...

    See docs.oracle.com/javase/8/docs/api/java/util/function/Consumer.html
Using a Parallel Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string.

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    boolean tryAdvance(Consumer<? super Result> action) {
        if (!mPhraseMatcher.find())
            return false;
        else {
            action.accept(new Result(
                mOffset + mPhraseMatcher.start()));
            return true;
        }
    }
    ...
}
```

See [docs.oracle.com/javase/8/docs/api/java/util/regex/Matcher.html#find](https://docs.oracle.com/javase/8/docs/api/java/util/regex/Matcher.html#find)
Using a Parallel Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string.

```java
class PhraseMatchSpliterator implements Spliterator<SearchResults> {
    ...
    boolean tryAdvance(Consumer<? super SearchResults> action) {
        if (!mPhraseMatcher.find())
            return false;
        else {
            action.accept(new SearchResults(mOffset + mPhraseMatcher.start()));
            return true;
        }
    }
    ...
```

Informe the streams framework to cease calling tryAdvance() if there's no match.
PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string.

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    boolean tryAdvance(Consumer<? super Result> action) {
        if (!mPhraseMatcher.find())
            return false;
        else {
            action.accept(new Result(mOffset + mPhraseMatcher.start()));
            return true;
        }
    }
    ...
}
```

If there is a match, then accept() keeps track of which index in the input string the match occurred.

See docs.oracle.com/javase/8/docs/api/java/util/function/Consumer.html#accept
Using a Parallel Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    ... 
    boolean tryAdvance(Consumer<? super Result> action) {
        if (!mPhraseMatcher.find())
            return false;

        else {
            action.accept(new Result
                (mOffset + mPhraseMatcher.start()));
            return true;
        }
    }
    ...
}
```

Inform the streams framework to continue calling tryAdvance()
Using a Parallel Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    Spliterator<Result> trySplit() { ... }

    int computeStartPos(int splitPos) { ... }

    int tryToUpdateSplitPos(int startPos, int splitPos) { ... }

    PhraseMatchSpliterator splitInput(int splitPos) { ... }
    ...
}
```

These methods are used for parallel streams
Using a Parallel Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    Spliterator<Result> trySplit() { ... }

    int computeStartPos(int splitPos) { ... }

    int tryToUpdateSplitPos(int startPos, int splitPos) { ... }

    PhraseMatchSpliterator splitInput(int splitPos) { ... }
    ...

    These methods are used for parallel streams

Note that there is no synchronization in any of these methods!!!
Using a Parallel Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string class PhraseMatchSpliterator implements Spliterator<Result> {
  ...
  Spliterator<Result> trySplit() {
    if (input is below minimum size) return null
    else {
      split input in 2 relatively even-sized chunks
      return a spliterator for "left chunk"
    }
  }
  ...
}

trySplit() attempts to split the input "evenly" so phrases can be matched in parallel
Using a Parallel Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    ...  
    Spliterator<Result> trySplit() {
        // Splits don’t need to be perfectly equal in order for the spliterator to be efficient
    }
```
PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string.

class PhraseMatchSpliterator implements Spliterator<Result> {

    Spliterator<Result> trySplit() {
        if (mInput.length() <= mMinSplitSize) return null;
        int startPos, splitPos;
        splitPos = mInput.length() / 2;

        if ((startPos = computeStartPos(splitPos)) < 0) return null;

        if ((splitPos = tryToUpdateSplitPos(startPos, splitPos)) < 0) return null;

        return splitInput(splitPos); ...

This code is heavily commented, so please check it out
PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string. The class PhraseMatchSpliterator implements Spliterator<Result>:

```
... {
    Spliterator<Result> trySplit() {
        if (mInput.length() <= mMinSplitSize) return null;
        int startPos, splitPos = mInput.length() / 2;

        if ((startPos = computeStartPos(splitPos)) < 0) return null;

        if ((splitPos = tryToUpdateSplitPos(startPos, splitPos)) < 0) return null;

        return splitInput(splitPos); ...
    }
}
```

Bail out if input is too small to split further.
Using a Parallel Spliterator in SearchStreamGang

“... Therefore, since brevity is the soul of” “wit, And tediousness the limbs and outward...”

```
class PhraseMatchSpliterator implements Spliterator<Result> {
    ...    // Initial guess at the split position
    Spliterator<Result> trySplit() {
        if (mInput.length() <= mMinSplitSize) return null;
        int startPos,
            splitPos = mInput.length() / 2;

        if (((startPos = computeStartPos(splitPos)) < 0) return null;

        if (((splitPos = tryToUpdateSplitPos(startPos, splitPos)) < 0)
            return null;

        return splitInput(splitPos); ...
    }
}
```
Using a Parallel Spliterator in SearchStreamGang

“... Therefore, since *brevity is the soul of* “*wit*, And tediousness the limbs and outward...”

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    Spliterator<Result> trySplit() {
        if (mInput.length() <= mMinSplitSize) return null;
        int startPos, splitPos = mInput.length() / 2;

        if ((startPos = computeStartPos(splitPos)) < 0) return null;

        if ((splitPos = tryToUpdateSplitPos(startPos, splitPos)) < 0)
            return null;

        return splitInput(splitPos); ...
    }
```

Initial guess at where to start the search
class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    int computeStartPos(int splitPos) {
        int phraseLength = mPhrase.length();
        int startPos = splitPos - phraseLength;
        if (startPos < 0 || phraseLength > splitPos)
            return -1;
        else
            return startPos;
    }
    
    Identify the position to start determining if a phrase spans the split position

"... Therefore, since brevity is the soul of” “wit, And tediousness the limbs and outward..."
Using a Parallel Spliterator in SearchStreamGang

"... Therefore, since brevity is the soul of” “wit, And tediousness the limbs and outward...

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    int computeStartPos(int splitPos) {
        int phraseLength = mPhrase.length();

        int startPos = splitPos - phraseLength;

        if (startPos < 0 || phraseLength > splitPos)
            return -1;
        else
            return startPos;
    }
}
```

"... Therefore, since brevity is the soul of” “wit, And tediousness the limbs and outward..."
Using a Parallel Spliterator in SearchStreamGang

“... Therefore, since brevity is the soul of” “wit, And tediousness the limbs and outward...”

```
class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    int computeStartPos(int splitPos) {
        int phraseLength = mPhrase.length();
        int startPos = splitPos - phraseLength;

        if (startPos < 0 || phraseLength > splitPos)
            return -1;
        else
            return startPos;
    }
```

Compute the initial startPos by subtracting the phrase length from the splitPos
Using a Parallel Spliterator in SearchStreamGang

“... Therefore, since brevity is the soul of” “wit, And tediousness the limbs and outward...”

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
  ...
  int computeStartPos(int splitPos) {
    int phraseLength = mPhrase.length();
    int startPos = splitPos - phraseLength;
    if (startPos < 0 || phraseLength > splitPos)
      return -1;
    else
      return startPos;
  }
}
```

Fail if phrase is too long for this input segment
class PhraseMatchSpliterator implements Spliterator<Result> {

    int computeStartPos(int splitPos) {

        int phraseLength = mPhrase.length();

        int startPos = splitPos - phraseLength;

        if (startPos < 0 || phraseLength > splitPos)
            return -1;
        else
            return startPos;
    }

    Return the computed start position

    ... Therefore, since brevity is the soul of” “wit, And tediousness the limbs and outward..."
Using a Parallel Spliterator in SearchStreamGang

class PhraseMatchSpliterator implements Spliterator<Result> {
    ...

    Spliterator<Result> trySplit() {
        if (mInput.length() <= mMinSplitSize) return null;
        int startPos,
            splitPos = mInput.length() / 2;

        if (((startPos = computeStartPos(splitPos)) < 0) return null;

        if (((splitPos = tryToUpdateSplitPos(startPos, splitPos)) < 0)
            return null;

        return splitInput(splitPos); ...

    }

    // Update splitPos if phrase spans the initial splitPos

“... Therefore, since brevity is the soul of wit” “, And tediousness the limbs and outward...”
Using a Parallel Spliterator in SearchStreamGang

“... Therefore, since brevity is the soul of” “wit, And tediousness the limbs and outward...”

class PhraseMatchSpliterator implements Spliterator<Result> {
...
    int tryToUpdateSplitPos(int startPos, int splitPos) {
        int endPos =
            splitPos + mPattern.toString().length();
        if (endPos >= mInput.length()) return -1;
        CharSequence substr =
            mInput.subSequence(startPos, endPos);
        Matcher pm = mPattern.matcher(substr);
        if (pm.find()) splitPos = startPos
            + pm.start() + pm.group().length();
        return splitPos;
    }
}
Using a Parallel Spliterator in SearchStreamGang

“... Therefore, since brevity is the soul of” “wit, And tediousness the limbs and outward...”

```
class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    int tryToUpdateSplitPos(int startPos, int splitPos) {
        int endPos =
            splitPos + mPattern.toString().length();
        if (endPos >= mInput.length()) return -1;
        CharSequence substr =
            mInput.subSequence(startPos, endPos);
        Matcher pm = mPattern.matcher(substr);
        if (pm.find()) splitPos = startPos
            + pm.start() + pm.group().length();
        return splitPos;
    }
```

Ensure phrase isn’t longer than the input string!
Using a Parallel Spliterator in SearchStreamGang

class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    int tryToUpdateSplitPos(int startPos, int splitPos) {
        int endPos =
            splitPos + mPattern.toString().length();
        if (endPos >= mInput.length()) return -1;
        CharSequence substr =
            mInput.substring(startPos, endPos);
        Matcher pm = mPattern.matcher(substr);
        if (pm.find()) splitPos = startPos
            + pm.start() + pm.group().length();
        return splitPos;
    }
}

"... Therefore, since brevity is the soul of" "wit, And tediousness the limbs and outward..."

Check to see if the phrase matches within the substring that span the initial splitPos

"brevity is the soul of wit"
Using a Parallel Spliterator in SearchStreamGang

"... Therefore, since brevity is the soul of wit" "And tediousness the limbs and outward..."

class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    int tryToUpdateSplitPos(int startPos, int splitPos) {
        int endPos =
            splitPos + mPattern.toString().length();
        if (endPos >= mInput.length()) return -1;
        CharSequence substr =
            mInput.subSequence(startPos, endPos);
        Matcher pm = mPattern.matcher(substr);
        if (pm.find()) splitPos = startPos
            + pm.start() + pm.group().length();
        return splitPos;
    }

    If there's a match update the splitPos to handle phrase spanning newlines
Using a Parallel Spliterator in SearchStreamGang

“... Therefore, since **brevity is the soul of wit**” “, And tediousness the limbs and outward...”

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    ... tryToUpdateSplitPos(int startPos, int splitPos) {
        int endPos =
            splitPos + mPattern.toString().length();
        if (endPos >= mInput.length()) return -1;
        CharSequence substr =
            mInput.subSequence(startPos, endPos);
        Matcher pm = mPattern.matcher(substr);
        if (pm.find()) splitPos = startPos
            + pm.start() + pm.group().length();
        return splitPos;
    }
```

Return the final `splitPos`
Using a Parallel Spliterator in SearchStreamGang

“... Therefore, since brevity is the soul of wit” “... And tediousness the limbs and outward…”

Left Hand Spliterator

Right Hand Spliterator

class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    Spliterator<Result> trySplit() {
        if (mInput.length() <= mMinSplitSize) return null;
        int startPos, splitPos,
            splitPos = mInput.length() / 2;

        if (startPos = computeStartPos(splitPos)) < 0) return null;

        if (splitPos = tryToUpdateSplitPos(startPos, splitPos)) < 0) return null;

        return splitInput(splitPos); ...
    }

    Create & return a new spliterator
Using a Parallel Spliterator in SearchStreamGang

"... Therefore, since brevity is the soul of wit" "And tediousness the limbs and outward..."

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    Splitterator<Result> splitInput(int splitPos) {
        CharSequence lhs =
            mInput.subSequence(0, splitPos);
        mInput = mInput.subSequence(splitPos, mInput.length());
        mPhraseMatcher = mPattern.matcher(mInput);
        ...
        return new PhraseMatchSpliterator(lhs, ...);
    ...
```
Using a Parallel Spliterator in SearchStreamGang

“... Therefore, since brevity is the soul of wit” “... And tediousness the limbs and outward...”

class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    Spliterator<Result> splitInput(int splitPos) {
        CharSequence lhs =
            mInput.subSequence(0, splitPos);
        mInput = mInput.subSequence(splitPos, mInput.length());
        mPhraseMatcher = mPattern.matcher(mInput);
        ...
        return new PhraseMatchSpliterator(lhs, ...); ...

    }

    Create a sub-sequence for the left-hand spliterator

}
Using a Parallel Spliterator in SearchStreamGang

"... Therefore, since brevity is the soul of wit" "... And tediousness the limbs and outward...

Left Hand Spliterator

Right Hand Spliterator

class PhraseMatchSpliterator implements Spliterator<Result> {
  ...
  Spliterator<Result> splitInput(int splitPos) {
    CharSequence lhs =
      mInput.subSequence(0, splitPos);
    mInput = mInput.subSequence(splitPos, mInput.length());
    mPhraseMatcher = mPattern.matcher(mInput);
    ...
    return new PhraseMatchSpliterator(lhs, ...); ...
  }

Update "this" to reflect changes to "right hand" portion of input
Using a Parallel Spliterator in SearchStreamGang

"... Therefore, since brevity is the soul of wit" "And tediousness the limbs and outward..."

**Left Hand Spliterator**

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
...
    Spliterator<Result> splitInput(int splitPos) {
        CharSequence lhs =
            mInput.subSequence(0, splitPos);
        mInput = mInput.subSequence(splitPos, mInput.length());
        mPhraseMatcher = mPattern.matcher(mInput);
        ...
        return new PhraseMatchSpliterator(lhs, ...); ...  
    }
}
```

**Right Hand Spliterator**

Return a Spliterator that handles "left hand" portion of input, while "this" object handles "right hand" portion of input

return new PhraseMatchSpliterator(lhs, ...); ...
Using a Parallel Spliterator in SearchStreamGang

- Java 8 streams framework processes all spliterator chunks for each input string in parallel in the common fork-join pool

This parallelism is in addition to parallelism of input string & phrase chunks!!
End of Java 8
SearchWithParallelSpliterator Example (Part 1)