Java SearchWithParallelSpliterator

Example: trySplit()

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

• Be aware of how a parallel spliterator can improve parallel stream performance
• Know the intent of—and fields in—the PhraseMatchSpliterator
• Recognize the PhraseMatchSpliterator constructor & tryAdvance() method implementation
• Understand the PhraseMatchSpliterator trySplit() method implementation

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

This method is used with the SearchWithParallelSpliterators class
Analysis of the PhraseMatch Spliterator trySplit() Method
Analysis of the PhraseMatchSpliterator trySplit() Method

- The streams framework uses trySplit() partition a work of Shakespeare into chunks that can be searched in parallel

```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
The streams framework uses `trySplit()` to partition a work of Shakespeare into chunks that can be searched in parallel.

```java
class PhraseMatchSpliterator implements Spliterator<Result> {
    ...
    Spliterator<Result> trySplit() { ... }
    int computeStartPos(int splitPos) { ... }
    int tryToUpdateSplitPos(int startPos, int splitPos) { ... }
    PhraseMatchSpliterator splitInput(int splitPos) { ... }
    ...
```

There is **no** synchronization in any of these methods!!!

These methods are used for parallel streams

There is *no* synchronization in any of these methods!!!
Analysis of the PhraseMatchSpliterator `trySplit()` Method

- The streams framework uses `trySplit()` partition a work of Shakespeare into chunks that can be searched in parallel.

```java
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.
The streams framework uses `trySplit()` to partition a work of Shakespeare into chunks that can be searched in parallel.

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

Splits don’t needn’t be perfectly equal in order for the spliterator to be efficient.
The streams framework uses `trySplit()` to partition a work of Shakespeare into chunks that can be searched in parallel.

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

This code is heavily commented, so please check it out.
Analysis of the PhraseMatchSpliterator trySplit() Method

- The streams framework uses trySplit() partition a work of Shakespeare into chunks that can be searched in parallel.

```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); ...  

        Bail out if input is too small to split further
```
Analysis of the PhraseMatchSpliterator trySplit() Method

"... 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); ...
    }

    // Method definitions...
}
```

Initial guess at the split position

splitPos
Analysis of the PhraseMatchSpliterator trySplit() Method

“... 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
```

$startPos$ $splitPos$
Analysis of the PhraseMatchSpliterator trySplit() Method

“... 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;
    }
}
```

Identify the position to start determining if a phrase spans the split position
Analysis of the PhraseMatchSpliterator trySplit() Method

"... 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;
  }
```

Store the length of the phrase
Analysis of the PhraseMatchSpliterator trySplit() Method

“... 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;
    }
}
```

Compute the initial startPos by subtracting the phrase length from the splitPos
Analysis of the PhraseMatchSpliterator trySplit() Method

“... 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...”

    startPos
    splitPos
Analysis of the PhraseMatchSpliterator trySplit() Method

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

```
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
Analysis of the PhraseMatchSpliterator trySplit() Method

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

```java
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;
    }
}
```

Don’t split a string across a phrase
Analysis of the PhraseMatchSpliterator `trySplit()` Method

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

```java
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;
    }
```

Set `endPos` to the very end of the input that could match the pattern....
Analysis of the PhraseMatchSpliterator trySplit() Method

“... 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!
Analysis of the PhraseMatchSpliterator trySplit() Method

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

```java
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;
    }
```

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

“brevity is the soul of wit”
Analysis of the PhraseMatchSpliterator trySplit() Method

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

```java
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
Analysis of the PhraseMatchSpliterator trySplit() Method

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

```java
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;
    }
}
```

Return the final `splitPos`
Analysis of the PhraseMatchSpliterator trySplit() Method

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

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

Create & return a new spliterator
Analysis of the PhraseMatchSpliterator trySplit() Method

“... 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);
        mOffset = splitPos;
        ...

        return new PhraseMatchSpliterator(lhs, ...); ...
    }

"… Therefore, since brevity is the soul of wit” “And tediousness the limbs and outward...”
Analysis of the PhraseMatchSpliterator trySplit() Method

```
... 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);
    mOffset = splitPos;
    ...
    return new PhraseMatchSpliterator(lhs, ...); ...
```

Create a sub-sequence for the left-hand spliterator
Analysis of the PhraseMatchSpliterator trySplit() Method

```
“... 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);
        mOffset = splitPos;
        ...

        Update “this” to reflect changes to “right hand” portion of input

        return new PhraseMatchSpliterator(lhs, ...); ...
    }
```

27
Analysis of the PhraseMatchSpliterator trySplit() Method

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

This spliterator handles "left hand" portion of input, while "this" object handles "right hand" portion of input.

```
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);
        mOffset = splitPos;
        ...

        return new PhraseMatchSpliterator(lhs, ...); ...
    }
}
```
Analysis of the PhraseMatchSpliterator trySplit() Method

- Java 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 SearchWith ParallelSpliterator Example: trySplit()