Overview of the Java Search
WithParallelSpliterator Case Study

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

• Understand how a parallel spliterator can improve parallel stream performance

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

45,000+ phrases
Learning Objectives in this Part of the Lesson

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

See “Java 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
<<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
Overview of SearchWithParallelSpliterator

• SearchWithParallelSpliterator uses parallel streams in three ways
  
  • Search chunks of input in parallel
  
  • Search chunks of phrases in parallel
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
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 aggressive parallelism strategy!
Overview of SearchWithParallelSpliterator

- The relative contribution of each parallel streams model is shown here:
  - Time for 38 strings = 462 ms (parallelSpliterator|parallelPhrases|parallelInput)
  - Time for 38 strings = 470 ms (sequentialSpliterator|parallelPhrases|parallelInput)
  - Time for 38 strings = 477 ms (sequentialSpliterator|parallelPhrases|sequentialInput)
  - Time for 38 strings = 490 ms (parallelSpliterator|parallelPhrases|sequentialInput)
  - Time for 38 strings = 498 ms (parallelSpliterator|sequentialPhrases|parallelInput)
  - Time for 38 strings = 510 ms (sequentialSpliterator|sequentialPhrases|parallelInput)
  - Time for 38 strings = 1326 ms (parallelSpliterator|sequentialPhrases|sequentialInput)
  - Time for 38 strings = 2463 ms (sequentialSpliterator|sequentialPhrases|sequentialInput)

See [github.com/douglascraigschmidt/LiveLessons/tree/master/SearchStreamSpliterator](https://github.com/douglascraigschmidt/LiveLessons/tree/master/SearchStreamSpliterator)
Overview of SearchWithParallelSpliterator

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

  Time for 38 strings = 462 ms \((\text{parallelSpliterator} | \text{parallelPhrases} | \text{parallelInput})\)
  Time for 38 strings = 470 ms \((\text{sequentialSpliterator} | \text{parallelPhrases} | \text{parallelInput})\)
  Time for 38 strings = 477 ms \((\text{sequentialSpliterator} | \text{parallelPhrases} | \text{sequentialInput})\)
  Time for 38 strings = 490 ms \((\text{parallelSpliterator} | \text{parallelPhrases} | \text{sequentialInput})\)
  Time for 38 strings = 498 ms \((\text{parallelSpliterator} | \text{sequentialPhrases} | \text{parallelInput})\)
  Time for 38 strings = 510 ms \((\text{sequentialSpliterator} | \text{sequentialPhrases} | \text{parallelInput})\)
  Time for 38 strings = 1326 ms \((\text{parallelSpliterator} | \text{sequentialPhrases} | \text{sequentialInput})\)
  Time for 38 strings = 2463 ms \((\text{sequentialSpliterator} | \text{sequentialPhrases} | \text{sequentialInput})\)

Tests conducted on a 2.6 GHz six-core Lenovo P52 with 64 Gbytes of RAM
Overview of SearchWithParallelSpliterator

- Longer input strings leverage the parallel spliterator even better:
  - Time for 2 strings = 452 ms (parallelSpliterator|parallelPhrases|parallelInput)
  - Time for 2 strings = 462 ms (sequentialSpliterator|parallelPhrases|parallelInput)
  - Time for 2 strings = 466 ms (sequentialSpliterator|parallelPhrases|sequentialInput)
  - Time for 2 strings = 478 ms (parallelSpliterator|parallelPhrases|sequentialInput)
  - Time for 2 strings = 788 ms (parallelSpliterator|sequentialPhrases|parallelInput)
  - Time for 2 strings = 1298 ms (sequentialSpliterator|sequentialPhrases|parallelInput)
  - Time for 2 strings = 1488 ms (parallelSpliterator|sequentialPhrases|sequentialInput)
  - Time for 2 strings = 2467 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

```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 `livelessons/streamgangs/SearchStreamGang.java`
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()));
}
```

*StreamSupport.stream()* creates a sequential or parallel stream via *PhraseMatchSpliterator*

See [docs.oracle.com/javase/8/docs/api/java/util/stream/StreamSupport.html#stream](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](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

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

We now focus in depth on the PhraseMatchSpliterator methods

See the rest of the lessons on "Java SearchWithParallelSpliterator Example"
End of Overview of the Java SearchWithParallelSpliterator Case Study