Java 8 SearchWithParallelStreams

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

- Know how Java 8 parallel streams are applied in `SearchWithParallelStreams`

Starting `SearchStreamGangTest`
- `PARALLEL_SPLITTERATOR` executed in 409 msecs
- `COMPLETABLE_FUTURES_INPUTS` executed in 426 msecs
- `COMPLETABLE_FUTURES_PHASES` executed in 427 msecs
- `PARALLEL_STREAMS` executed in 437 msecs
- `PARALLEL_STREAM_PHASES` executed in 440 msecs
- `RXJAVA_PHASES` executed in 485 msecs
- `PARALLEL_STREAM_INPUTS` executed in 802 msecs
- `RXJAVA_INPUTS` executed in 866 msecs
- `SEQUENTIAL_LOOPS` executed in 1638 msecs
- `SEQUENTIAL_STREAM` executed in 1958 msecs

Ending `SearchStreamGangTest`

See `github.com/douglascraigschmidt/LiveLessons/tree/master/SearchStreamGang`
Applying Parallel Streams to SearchStreamGang
Applying Parallel Streams to SearchStreamGang

- We focus on parallel streams in `processStream()` & `processInput()` from `SearchWithParallelStreams`

<table>
<thead>
<tr>
<th>&lt;&lt;Java Class&gt;&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>SearchWithParallelStreams</td>
</tr>
<tr>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>processStream(): List/List&lt;SearchResults&gt;</td>
</tr>
<tr>
<td>processInput(CharSequence): List&lt;SearchResults&gt;</td>
</tr>
</tbody>
</table>

Applying Parallel Streams to SearchStreamGang

- We focus on parallel streams in `processStream()` & `processInput()` from `SearchWithParallelStreams`

```java
getInput().parallelStream().map(this::processInput).collect(toList());
```

```java
return mPhrasesToFind.parallelStream().map(phrase -> searchForPhrase(phrase, input, title, false)).filter(not(SearchResults::isEmpty)).collect(toList());
```

See `SearchStreamGang/src/main/java/livelessons/streamgangs/SearchWithParallelStreams.java`
Applying Parallel Streams to SearchStreamGang

- We focus on parallel streams in `processStream()` & `processInput()` from `SearchWithParallelStreams`.

```java
getInput()
  .parallelStream()
  .map(this::processInput)
  .collect(toList());
```

```java
return mPhrasesToFind
  .parallelStream()
  .map(phrase -> searchForPhrase(phrase, input, title, false))
  .filter(not(SearchResults::isEmpty))
  .collect(toList());
```

i.e., the map(), filter(), & collect() aggregate operations
Applying Parallel Streams to SearchStreamGang

- We focus on parallel streams in `processStream()` & `processInput()` from `SearchWithParallelStreams`

  - **`processStream()`**
    - Uses a parallel stream to search a list of input strings within the common fork-join pool of worker threads

  - **`processInput()`**

---

Each input string contains one work of Shakespeare (e.g., *Hamlet*, *MacBeth*, etc.)
We focus on parallel streams in `processStream()` & `processInput()` from `SearchWithParallelStreams`:

- **processStream()**
- **processInput()**

  - Uses a parallel stream to search each input string & locate all occurrences of phases within the common fork-join pool of worker threads.
Visualizing processStream() & processInput()
processStream() searches a list of input strings in parallel

List <String>

Input a list of input strings

parallelStream()
Visualizing `processStream()` & `processInput()`

- `processStream()` searches a list of input strings in parallel

Convert collection to a parallel stream, i.e., substreams with chunks of input strings
Visualizing `processStream()` & `processInput()`

- `processStream()` searches a list of input strings in parallel

```
Output a stream of input strings
```

List

```
_stream<String>
```

Stream

```
_stream<String>
```

Input Strings to Search

```
[Input Strings]
```

**parallelStream()**

Chunks of input strings are processed in parallel on separate threads/cores
Visualizing processStream() & processInput()

- processStream() searches a list of input strings in parallel

```
List <String>…
```

```
Stream <String>…
```

```
Input a stream of input strings
```

```
parallelStream()
```

```
map(this::processInput)
```

```
Input Strings to Search
```

13
• `processStream()` searches a list of input strings in parallel

**Call `processInput()` to search for phrases in an input string in parallel**
processStream() searches a list of input strings in parallel. It outputs a stream of lists of search results.

Visualizing processStream() & processInput()

- list of input strings
- output stream of lists of search results
- parallelStream()
- map(this::processInput)
Visualizing `processStream()` & `processInput()`

- `processStream()` searches a list of input strings in parallel

```java
List<String> inputStrings = ...;
Stream<String> parallelStream = inputStrings.parallelStream();
Stream<List<SearchResults>> processInput = parallelStream.map(this::processInput);
List<List<SearchResults>> results = processInput.collect(toList());
```

*Input a stream of lists of search results*
Visualizing `processStream()` & `processInput()`

- `processStream()` searches a list of input strings in parallel

```

List<String>
```

Trigger intermediate operation processing to run on multiple worker threads & cores
Visualizing `processStream()` & `processInput()`

- `processStream()` searches a list of input strings in parallel

```
List<String>…
```

```
Stream<String>…
```

```
Stream<List<SearchResults>>…
```

```
List<SearchResults>…
```

- Return a list of lists of search results based on “encounter order”
• processInput() finds phrases in an input string in parallel

**Input a list of phrases to find**
processInput() finds phrases in an input string in parallel

Visualizing `processStream()` & `processInput()`

Convert collection to a parallel stream, i.e., substreams with chunks of phrases
• `processInput()` finds phrases in an input string in parallel

**Output a stream of phrases to find**

List `<String>`

Stream `<String>`

Different chunks of phrases are processed in parallel on multiple worker threads & cores
Visualizing `processStream()` & `processInput()`

- `processInput()` finds phrases in an input string in parallel

```
Input a stream of phrases to find
```

```
List <String>
```

```
Stream <String>
```

```
parallelStream()
```

```
map(phrase -> searchForPhrase(...))
```

```
Search Phrases
```

```
Hamlet
```

45,000+ phrases

22
Visualizing `processStream()` & `processInput()`

- `processInput()` finds phrases in an input string in parallel

List `<String>` → parallelStream() → map(phrase -> searchForPhrase(…)) → Stream `<String>`

Search for phrases in each input string in parallel
• `processInput()` finds phrases in an input string in parallel

Output a stream of search results

List `<String>`

Stream `<String>`

Stream `<SearchResults>`

Parallel `processStream()` & `processInput()`

List `<String>`

Stream `<String>`

Stream `<SearchResults>`

`parallelStream()`

`map(phrase -> searchForPhrase(...))`
processInput() finds phrases in an input string in parallel.

List <String>

Stream <String>

Stream <SearchResults>

Input a stream of search results

parallelStream()

map(phrase -> searchForPhrase(...))

filter(not(SearchResults::isEmpty))
Visualizing `processStream()` & `processInput()`

- `processInput()` finds phrases in an input string in parallel.

List `<String>

Stream `<String>

Stream `<SearchResults>

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

Remove empty search results from substreams in parallel.
• processInput() finds phrases in an input string in parallel

Output a stream of non-empty search results

List <String>

Stream <String>

Stream <SearchResults>

Stream <SearchResults>

parallelStream()

map(phrase -> searchForPhrase(…))

filter(not/SearchResults::isEmpty)
processInput() finds phrases in an input string in parallel.

**Input a stream of non-empty search results**

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

Stream processing:

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

- `processInput()` finds phrases in an input string in parallel

```
List <String> processInput() finds phrases in an input string in parallel

Stream <String> processStream() & processInput()

Stream <SearchResults> List <String> map(phrase -> searchForPhrase(…))

Stream <SearchResults> filter(not(SearchResults::isEmpty))

List <String> collect(toList())
```

Trigger intermediate operation processing to run on multiple threads/cores
processInput() finds phrases in an input string in parallel.

```
List <String> processInput() finds phrases in an input string in parallel
```

```
Stream <String> processStream() & processInput()
```

```
List <String> map(phrase -> searchForPhrase(…))
```

```
Stream <SearchResults> filter(not(SearchResults::isEmpty))
```

```
Stream <SearchResults> collect(toList())
```

```
List <SearchResults> parallelStream()
```

```
Search Phrases
```

```
Return a list of search results in the originating thread based on “encounter order”
```
• Note that the actual processing of parallel streams differs from this visualization.

List `<String>`

Stream `<String>`

Stream `<SearchResults>`

Stream `<SearchResults>`

List `<SearchResults>`

See www.ibm.com/developerworks/library/j-java-streams-3-brian-goetz
Implementing `processStream()` as a Parallel Stream
Implementing processStream() as a Parallel Stream

- Parallel processStream() has one minuscule change wrt the sequential version

```java
protected List<List<SearchResults>> processStream() {
    List<CharSequence> inputList =
    getInput();

    return getInput()
        .parallelStream()
        .map(this::processInput)
        .collect(toList());
}
```
Implementing `processStream()` as a Parallel Stream

- Parallel `processStream()` has one minuscule change wrt the sequential version

```java
protected List<List<SearchResults>> processStream() {
    List<CharSequence> inputList = getInput();

    return getInput().
        parallelStream()
        .map(this::processInput)
        .collect(toList());
}
```

*Uses an ArrayList spliterator to create a parallel stream that searches an arraylist of input strings in multiple worker threads*
Implementing `processStream()` as a Parallel Stream

- Parallel `processStream()` has one minuscule change wrt the sequential version

```java
protected List<List<SearchResults>> processStream() {
    List<CharSequence> inputList = getInput();
    return getInput()
        .parallelStream()
        .map(this::processInput)
        .collect(toList());
}
```

“Chunks” of input strings are processed in parallel in the common fork-join pool
Implementing `processStream()` as a Parallel Stream

- Parallel `processStream()` has one minuscule change wrt the sequential version.

```java
protected List<List<SearchResults>> processStream() {
    List<CharSequence> inputList =
        getInput();

    return getInput()
        .parallelStream()
        .map(this::processInput)
        .collect(toList());
}
```

Searches a given input string to locate all occurrences of phases.
Implementing `processStream()` as a Parallel Stream

- Parallel `processStream()` has one minuscule change wrt the sequential version

```java
protected List<List<SearchResults>> processStream() {
    List<CharSequence> inputList = getInput();

    return getInput()
    .parallelStream()
    .map(this::processInput)
    .collect(toList());
}
```

Collectors.toList() returns a non-concurrent collector that obeys encounter order

Trigger intermediate operation processing & merge partial results into a single list of lists
Implementing `processInput()` as a Parallel Stream
Implementing `processInput()` as a Parallel Stream

Likewise, this `processInput()` implementation 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, false))
        .filter(not(SearchResults::isEmpty))
        .collect(toList());
    return results;
}
```
Implementing `processInput()` as a Parallel Stream

- Likewise, this `processInput()` implementation 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, false))
        .filter(not(SearchResults::isEmpty))
        .collect(toList());
    return results;
}
```

Create a parallel stream that searches the given input string to locate all occurrences of phases.
Implementing processInput() as a Parallel Stream

- Likewise, this processInput() implementation 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, false))
        .filter(not(SearchResults::isEmpty))
        .collect(toList());
    return results;
}
```

The PhraseMatchSpliterator breaks the input into “chunks” that are processed sequentially.

See docs.oracle.com/javase/8/docs/api/java/util/Spliterator.html
Implementing `processInput()` as a Parallel Stream

- Likewise, this `processInput()` implementation has just one minuscule change

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

    List<SearchResults> results =
        .parallelStream()
        .map(phase ->
            searchForPhrase(phase, 
                  .filter(not(SearchResults::
                        .collect(toList()));
    return results;
}
```

“Chunks” of phrases are processed in parallel in the common fork-join pool
Likewise, this processInput() implementation has just one minuscule change:

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

    List<SearchResults> results =
        .parallelStream()
        .map(phase ->
            searchForPhrase(phase, input,
                .filter(not(SearchResults::isEmpty))
            )
        )
        .collect(toList());
    return results;
}
```

Implementing `processInput()` as a Parallel Stream

- Trigger intermediate operation processing & merge partial results into a single list.
End of Java 8
SearchWithParallelStreams
Example (Part 1)