

Java SearchWithParallelSplitterator

Example: Evaluating Pros & Cons

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 `tryAdvance()` fields in the `PhraseMatchSpliterator`
- Recognize the `PhraseMatchSpliterator` constructor & `tryAdvance()` method implementation
- Understand the `PhraseMatchSpliterator` `trySplit()` method implementation
- Understand the pros & cons of the `SearchWithParallelSpliterator` class



<<Java Class>>

SearchWithParallelSpliterator

◆ `processStream():List<List<SearchResults>>`

■ `processInput(CharSequence):List<SearchResults>`

Pros of the SearchWith ParallelSpliterator Class

Pros of the SearchWithParallelSpliterator Class

- This example shows how a parallel spliterator can help transparently improve program performance

Input Strings to Search



Search Phrases



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

Pros of the SearchWithParallelSpliterator Class

- This example shows how a parallel spliterator can help transparently improve program performance

Input Strings to Search



Search Phrases



Starting SearchStreamGangTest

PARALLEL SPLITERATOR 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

Tests conducted on a 2.7GHz quad-core Lenovo P50 with 32 Gbytes of RAM

Pros of the SearchWithParallelSpliterator Class

- This example shows how a parallel spliterator can help transparently improve program performance

Input Strings to Search



Search Phrases



Starting SearchStreamGangTest

PARALLEL SPLITERATOR executed in 369 msec

PARALLEL STREAMS executed in 373 msec

COMPLETABLE_FUTURES_INPUTS executed in 377 msec

COMPLETABLE_FUTURES_PHASES executed in 383

PARALLEL_STREAM_PHASES executed in 385 msecs

RXJAVA_PHASES executed in 434 msec

PARALLEL_STREAM_INPUTS executed in 757 msecs

RXJAVA_INPUTS executed in 774 msecs

SEQUENTIAL LOOPS executed in 1485 msec

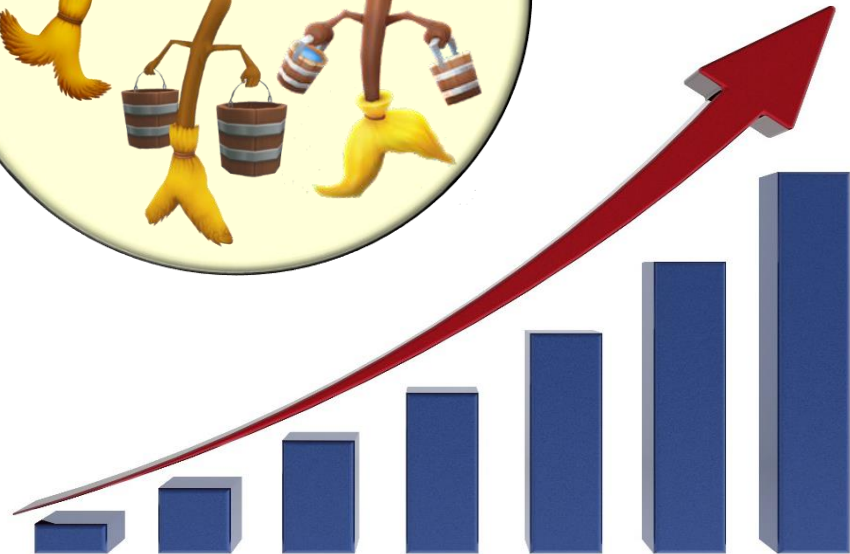
SEQUENTIAL STREAM executed in 1578 msec

Ending SearchStreamGangTest

Tests conducted on a 2.9GHz quad-core MacBook Pro with 16 Gbytes of RAM

Pros of the SearchWithParallelSpliterator Class

- This example shows how a parallel spliterator can help transparently improve program performance
- These speedups occur since the granularity of parallelism is finer & thus better able to leverage available cores

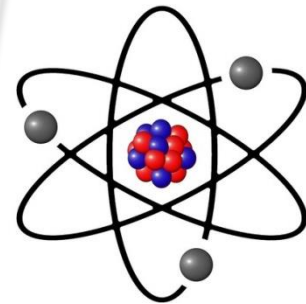


See docs.oracle.com/javase/tutorial/collections/streams/parallelism.html

Pros of the SearchWithParallelSpliterator Class

- This example also shows that the difference between using sequential vs parallel spliterator can be minuscule!

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



Switching this boolean from "false" to "true" controls whether the spliterator runs sequentially or in parallel

Pros of the SearchWithParallelSpliterator Class

- This example also shows that the difference between using sequential vs parallel spliterator can be minuscule!

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



Of course, it took non-trivial time/effort to create PhraseMatchSpliterator..

Cons of the SearchWith ParallelSpliterator Class

Cons of the SearchWithParallelSpliterator Class

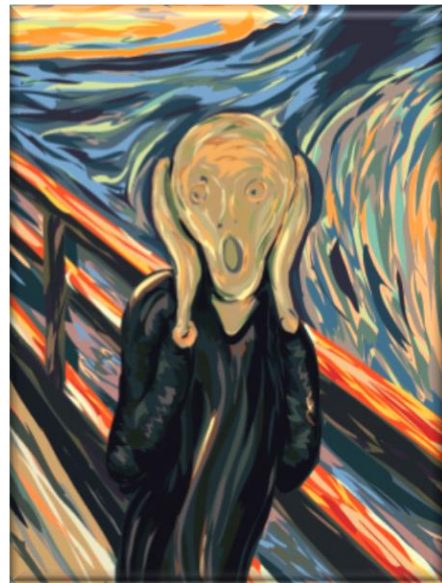
- The parallel-related portions of PhraseMatchSpliterator are *much* more complicated to program than the sequential-related portions...

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

    int computeStartPos(int splitPos) { ... }

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

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



Cons of the SearchWithParallelSpliterator Class

- The parallel-related portions of PhraseMatchSpliterator are *much* more complicated to program than the sequential-related portions...

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

    int computeStartPos(int splitPos) { ... }

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

Must split carefully..

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



JUnit tests are extremely useful..

Cons of the SearchWithParallelSpliterator Class

- The parallel-related portions of PhraseMatchSpliterator are *much* more complicated to program than the sequential-related portions...

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

    int computeStartPos(int splitPos) { ...

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

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



Writing the parallel spliterator took longer than writing the rest of the program!

End of Java SearchWith ParallelSpliterator Example: Evaluating Pros & Cons