Evaluating the Java SearchWith ParallelSpliterator Case Study

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

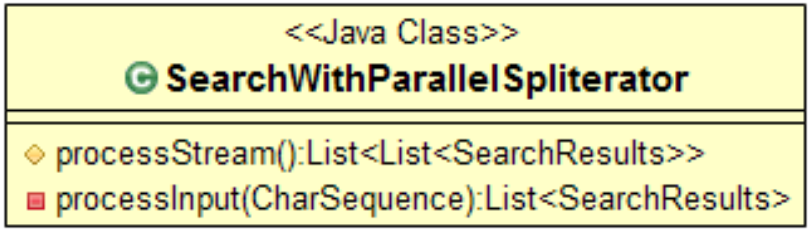
Institute for Software Integrated Systems

Vanderbilt University

Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

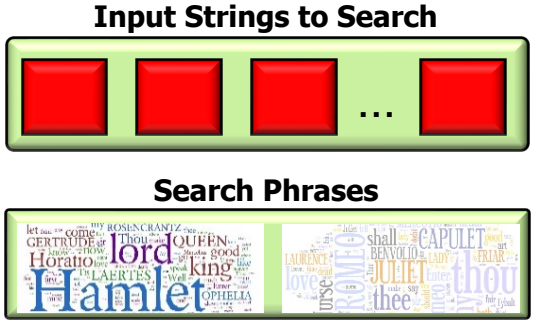
- Be aware of how a parallel spliterator can improve parallel stream performance
- Know the intent of—& fields in—the PhraseMatchSpliterator
- Recognize the PhraseMatchSpliterator constructor & tryAdvance() method implementation
- Understand the PhraseMatchSpliterator trySplit() method implementation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



Pros of the SearchWith ParallelSpliterator Class

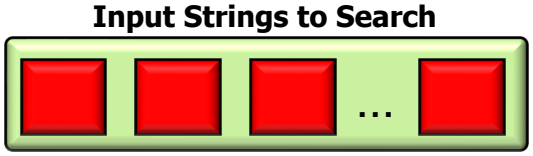
Pros of the SearchWithParallelSpliterator Class

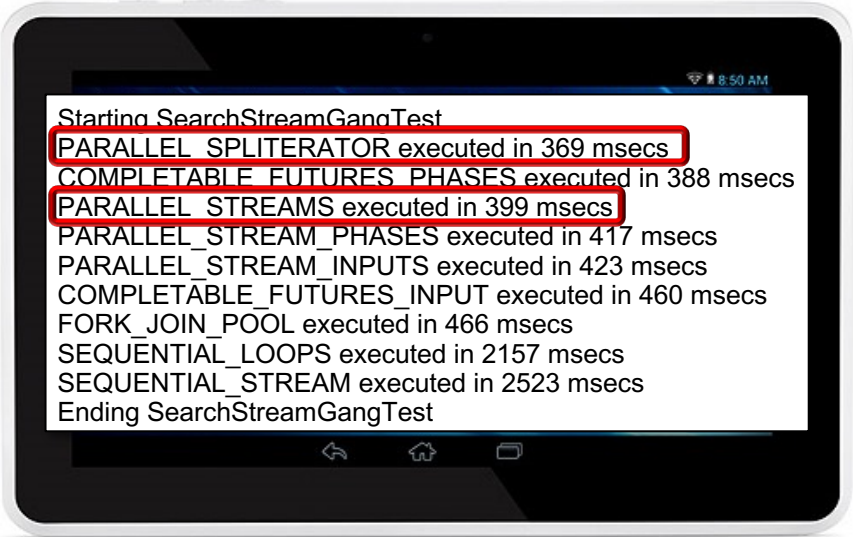
- This example shows how a parallel spliterator can help transparently improve program performance



Pros of the SearchWithParallelSpliterator Class

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

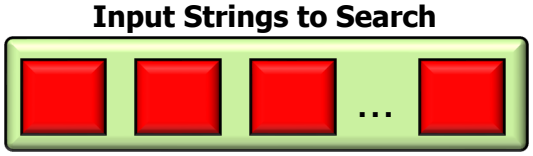


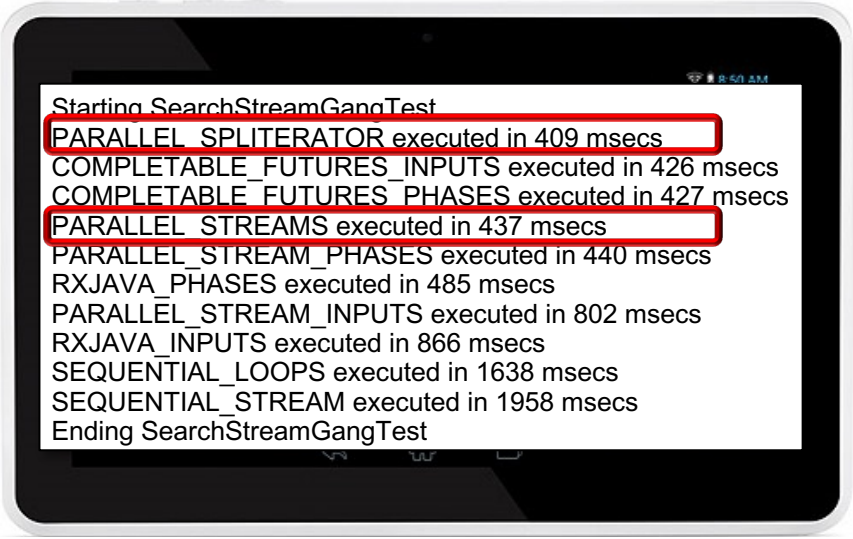


Tests conducted on a 3.2GHz 10-core MacBook Pro with 64 Gbytes of RAM

Pros of the SearchWithParallelSpliterator Class

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

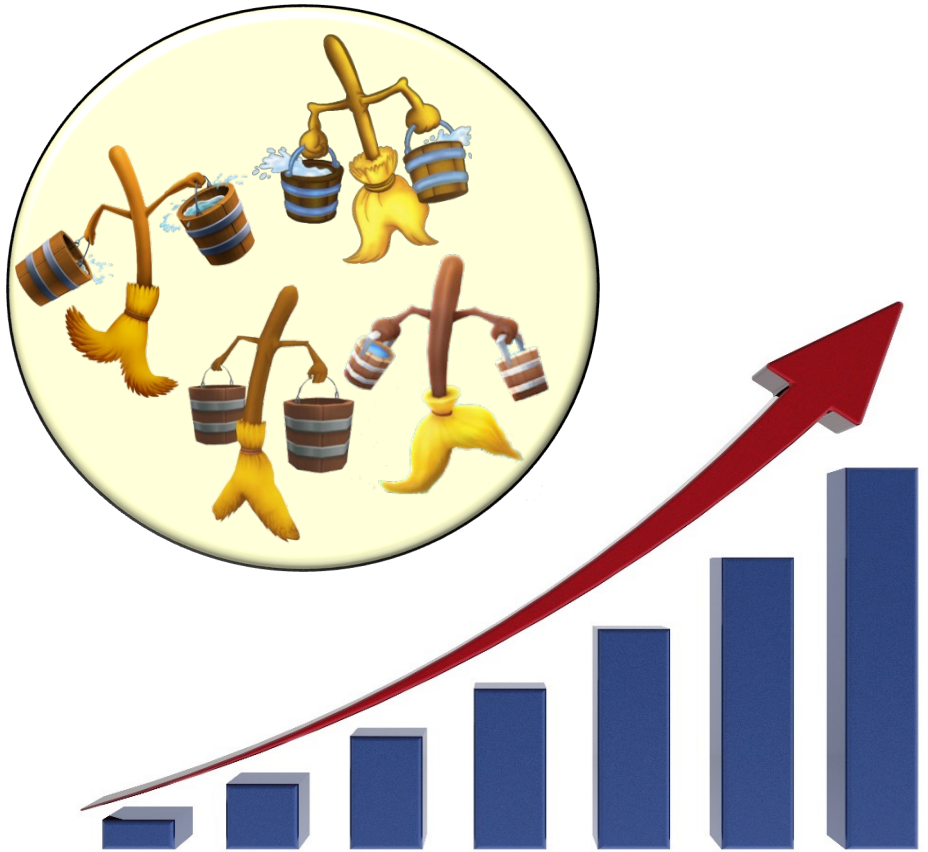




Tests conducted on a 2.7GHz quad-core Lenovo P50 with 32 Gbytes of RAM

Pros of the SearchWithParallelSpliterator Class

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

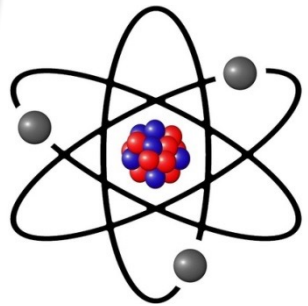


See docs.oracle.com/javase/tutorial/collections/streams/parallelism.html

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

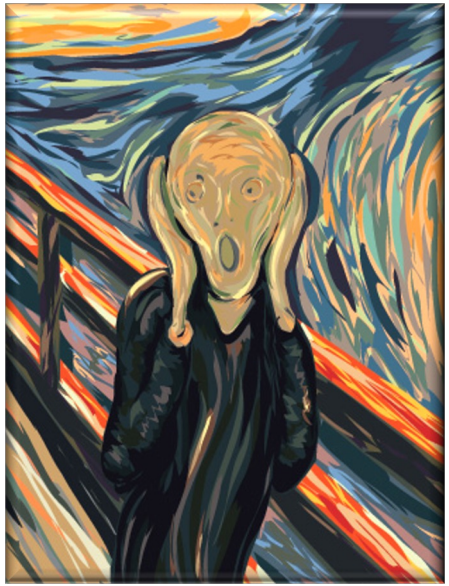
- The parallel-related portions of PhraseMatchSpliterator are *much* more complicated to program than the sequential-related portions...

```
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)
    { ... }

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

Must split carefully..

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 Evaluating the Java SearchWithParallelSpliterator Case Study