

Understand Java Parallel Streams Internals: Demo'ing Spliterator Performance

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 parallel stream internals, e.g.
 - Know what can change & what can't
 - Partition a data source into "chunks"
 - Know the impact of different Java collections on the performance of different spliterators

```
Starting spliterator tests for 1000 words..  
..printing results
```

```
17 msec: ArrayList parallel
```

```
19 msec: LinkedList parallel
```

```
Starting spliterator tests for 10000 words..  
..printing results
```

```
88 msec: LinkedList parallel
```

```
90 msec: ArrayList parallel
```

```
Starting spliterator tests for 100000 words..  
..printing results
```

```
599 msec: ArrayList parallel
```

```
701 msec: LinkedList parallel
```

```
Starting spliterator tests for 883311 words..  
..printing results
```

```
5718 msec: ArrayList parallel
```

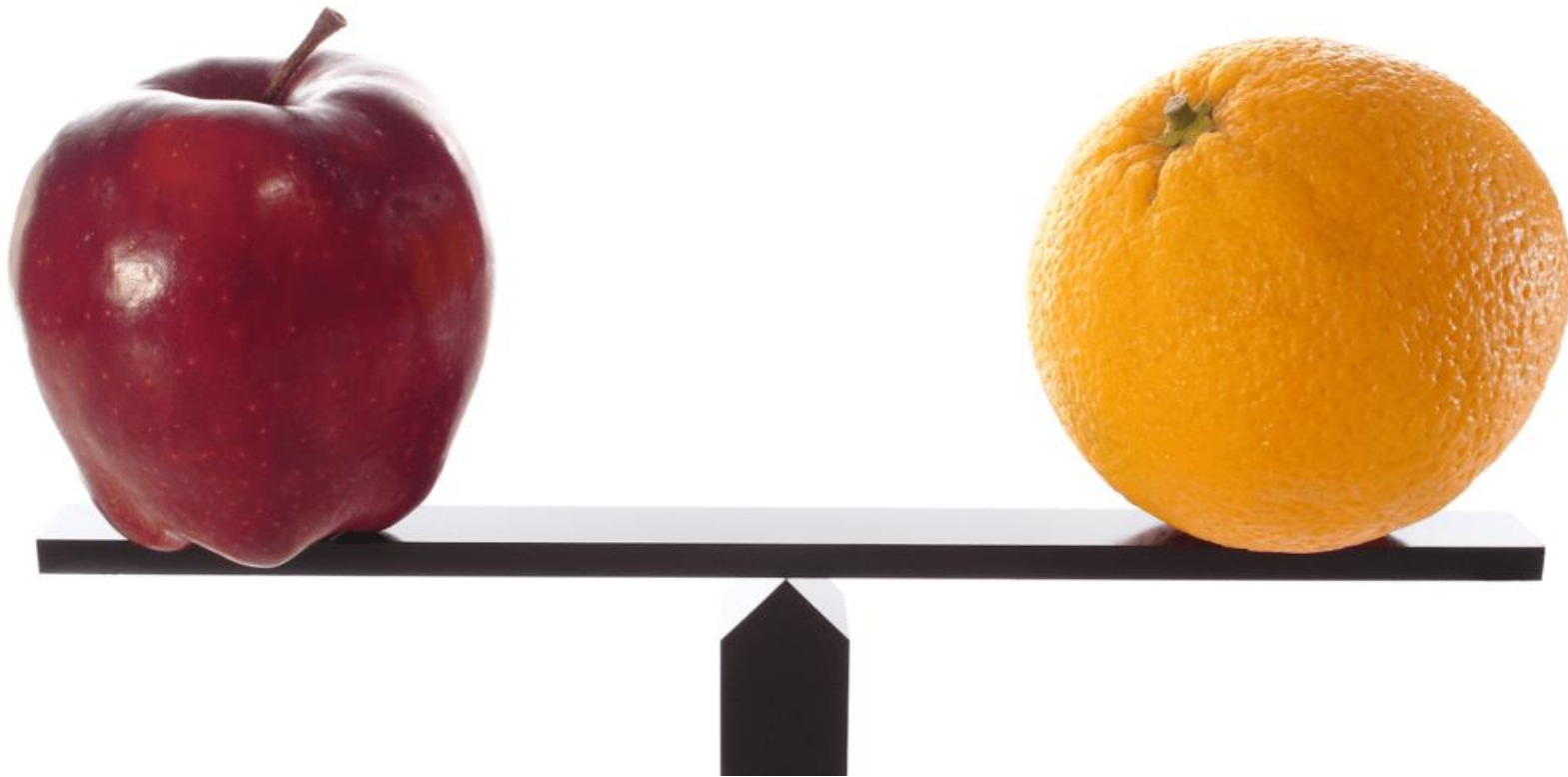
```
31226 msec: LinkedList parallel
```

See github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex14

Demonstrating Spliterator Performance

Demonstrating Splitter Performance

- Spliterators for ArrayList & LinkedList partition data quite differently



See earlier lesson on "*Java Parallel Streams Internals: Partitioning*"

Demonstrating Splitter Performance

- Spliterators for ArrayList & LinkedList partition data quite differently

```
ArrayListSpliterator<E> trySplit() {  
    int hi = getFence(), lo = index, mid = (lo + hi) >>> 1;  
    // divide range in half unless too small  
    return lo >= mid ? null : new ArrayListSpliterator<E>  
                                (list, lo, index = mid, ...);  
}
```

ArrayList's spliterator splits evenly & efficiently (e.g., doesn't copy data)



See openjdk/8u40-b25/java/util/ArrayList.java

Demonstrating Splitter Performance

- Spliterators for ArrayList & LinkedList partition data quite differently

```
Spliterator<E> trySplit() { ...  
    int n = batch + BATCH_UNIT, j = 0; Object[] a = new Object[n];  
    do { a[j++] = p.item; }  
    while ((p = p.next) != null && j < n); ...  
    return Spliterators.spliterator(a, 0, j, Spliterator.ORDERED);  
}
```

LinkedList's spliterator does not split evenly & efficiently (e.g., it copies data)



See openjdk/8u40-b25/java/util/LinkedList.java

Demonstrating Splitter Performance

- This demo program shows the performance difference of parallel spliterators for ArrayList & LinkedList when processing the complete works of Shakespeare

```
void timeParallelStreamUppercase(String testName,
                                List<CharSequence> words) {
    ...
    List<String> list = new ArrayList<>();

    for (int i = 0; i < sMAX_ITERATIONS; i++)
        list
            .addAll(words
                .parallelStream()
                .map(charSeq ->
                    charSeq.toString().toUpperCase())
                .collect(toList()))); ...
```

Demonstrating Splitter Performance

- This demo program shows the performance difference of parallel spliterators for ArrayList & LinkedList when processing the complete works of Shakespeare

```
void timeParallelStreamUppercase(String testName,  
                                List<CharSequence> words) {  
    ...  
    List<String> list = new ArrayList<>();  
  
    for (int i = 0; i < sMAX_ITERATIONS; i++)  
        list  
            .addAll(words  
                    .parallelStream()  
                    .map(charSeq ->  
                        charSeq.toString().toUpperCase())  
                    .collect(toList()))); ...
```

*The words param is passed
an ArrayList & a LinkedList*

Demonstrating Spliter Performance

- This demo program shows the performance difference of parallel spliterators for ArrayList & LinkedList when processing the complete works of Shakespeare

```
void timeParallelStreamUppercase(String testName,  
                                List<CharSequence> words) {  
    ...  
    List<String> list = new ArrayList<>();  
  
    for (int i = 0; i < sMAX_ITERATIONS; i++)  
        list  
            .addAll(words  
                    .parallelStream()  
                    .map(charSeq ->  
                        charSeq.toString().toUpperCase())  
                    .collect(toList()))); ...
```

*Split & uppercase
the word list via a
parallel spliterator*

Demonstrating Splitter Performance

- Results show splitter differences become more significant as input grows

Starting splitter tests for 1000 words....printing results

17 msecs: ArrayList parallel

19 msecs: LinkedList parallel

Starting splitter tests for 10000 words....printing results

88 msecs: ArrayList parallel

90 msecs: LinkedList parallel

Starting splitter tests for 100000 words....printing results

599 msecs: ArrayList parallel

701 msecs: LinkedList parallel

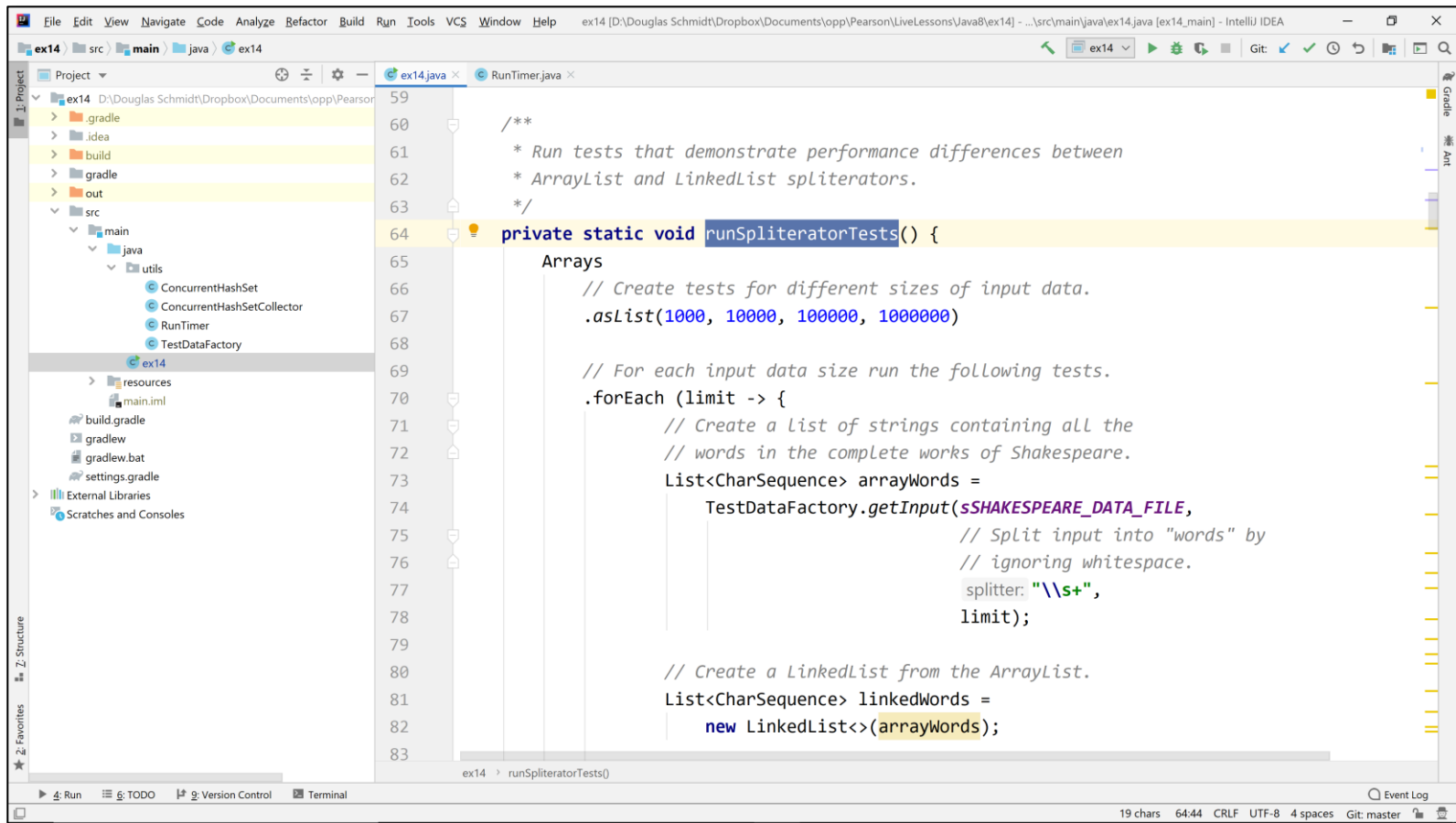
Starting splitter tests for 883311 words....printing results

5718 msecs: ArrayList parallel

31226 msecs: LinkedList parallel

See upcoming lessons on "*When [Not] to Use Parallel Streams*"

Demonstrating Spliterator Performance



```
59
60 /**
61  * Run tests that demonstrate performance differences between
62  * ArrayList and LinkedList spliterators.
63  */
64 private static void runSpliteratorTests() {
65     Arrays
66     // Create tests for different sizes of input data.
67     .asList(1000, 10000, 100000, 1000000)
68
69     // For each input data size run the following tests.
70     .forEach (limit -> {
71         // Create a list of strings containing all the
72         // words in the complete works of Shakespeare.
73         List<CharSequence> arrayWords =
74             TestDataFactory.getInput(SSHAKESPEARE_DATA_FILE,
75                                     // Split input into "words" by
76                                     // ignoring whitespace.
77                                     splitter: "\\s+",
78                                     limit);
79
80         // Create a LinkedList from the ArrayList.
81         List<CharSequence> linkedWords =
82             new LinkedList<>(arrayWords);
83     })
```

See github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex14

End of Understand Java Parallel Streams Internals: Demo'ing Splitter Performance