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 10000 words..
..printing results
  1 msecs: LinkedList sequential
  1 msecs: ArrayList sequential
  7 msecs: ArrayList parallel
  12 msecs: LinkedList parallel

Starting spliterator tests for 100000 words..
..printing results
  3 msecs: ArrayList parallel
  5 msecs: ArrayList sequential
  6 msecs: LinkedList sequential
  19 msecs: LinkedList parallel

Starting spliterator tests for 883311 words..
..printing results
  12 msecs: ArrayList parallel
  14 msecs: LinkedList parallel
  38 msecs: LinkedList sequential
  43 msecs: ArrayList sequential

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex14](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex14)
Demonstrating Spliterator Performance
Demonstrating Spliterator Performance

- Spliterators for ArrayList & LinkedList partition data quite differently

See earlier lesson on “Java Parallel Streams Internals: Partitioning”
Demonstrating Spliterator Performance

- Spliterators for ArrayList & LinkedList partition data quite differently

```java
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](openjdk/8u40-b25/java/util/ArrayList.java)
Demonstrating Spliterator Performance

- Spliterators for ArrayList & LinkedList partition data quite differently

```java
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 Spliterator Performance

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

```java
void timeStreamUppercase
  (String testName, List<String> words, boolean parallel) {
    ...
    List<String> list = new ArrayList<>();

    IntStream.range(0, sMAX_ITERATIONS).forEach((i) -> list
      .addAll(StreamSupport
        .stream(words.spliterator(),
              parallel)
        .map(string -> string.toUpperCase())
        .toList()));
  ...
```
This demo program shows the performance difference of parallel spliterators for ArrayList & LinkedList when processing the complete works of Shakespeare.

```java
void timeStreamUppercase
    (String testName, List<String> words, boolean parallel) {
        ...
        List<String> list = new ArrayList<>();
        IntStream.range(0, sMAX_ITERATIONS).forEach((i) -> list
            .addAll(StreamSupport
                .stream(words.spliterator(),
                    parallel)
                .map(string -> string.toUpperCase())
                .toList()));
    ...
```
void timeStreamUppercase
   (String testName, List<String> words, boolean parallel) {
   ...
   List<String> list = new ArrayList<>();
   ...
   IntStream.range(0, sMAX_ITERATIONS).forEach((i) -> list
            .addAll(StreamSupport
                    .stream(words.spliterator(),
                           parallel)
                    .map(string -> string.toUpperCase())
                    .toList()));
   ...

Demonstrating Spliterator Performance

• This demo program shows the performance difference of parallel spliterators for ArrayList & LinkedList when processing the complete works of Shakespeare.
Demonstrating Spliterator Performance

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

```java
void timeStreamUppercase
    (String testName, List<String> words, boolean parallel) {
    ...
    List<String> list = new ArrayList<>();

    IntStream.range(0, sMAX_ITERATIONS).forEach((i) -> list
        .addAll(StreamSupport
              .stream(words.spliterator(),
                     parallel)
              .map(string -> string.toUpperCase())
              .toList());
    ...
```

Conditionally select a parallel or sequential spliterator
Results show spliterator differences become more significant as input grows.

Starting spliterator tests for 100000 words..
..printing results
 3 msecs: ArrayList parallel
 5 msecs: ArrayList sequential
 6 msecs: LinkedList sequential
19 msecs: LinkedList parallel

Starting spliterator tests for 883311 words..
..printing results
12 msecs: ArrayList parallel
14 msecs: LinkedList parallel
38 msecs: LinkedList sequential
43 msecs: ArrayList sequential

See upcoming lessons on “When [Not] to Use Parallel Streams”
Demonstrating Spliterator Performance

```java
/**
 * Run tests that demonstrate performance differences between
 * ArrayList and LinkedList splitters.
 */

private static void runSplitterTests() {

    Arrays

    // Create tests for different sizes of input data.
    .aslist(1000, 10000, 100000, 1000000)

    // For each input data size run the following tests.
    .forEach(limit -> {
        // Create a list of strings containing all the
        // words in the complete works of Shakespeare.
        List<CharSequence> arrayWords =
            TestDataFactory.getInput(sSHakespeare_DATA_FILE,
                // Split input into "words" by
                // ignoring whitespace.
                splitter: "\s+",
                limit);

        // Create a LinkedList from the ArrayList.
        List<CharSequence> linkedWords =
            new LinkedList<>(arrayWords);
    });
```
End of Java Parallel Streams Internals: Demo’ing Spliterator Performance