Java Parallel Streams Internals: Overcoming Limitations with flatMap() in Parallel Streams

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"
  - Process chunks in parallel via the common fork-join pool
  - Configure the Java parallel stream common fork-join pool
  - Recognize how to overcome limitations with flatMap() in parallel streams

```
var result =
  generateOuterStream
    (Options.instance()
     .iterations())

 .flatMap(...::innerStream)

 .anyMatch(...);
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex35

Limitations with flatMap() in Parallel Streams

Limitations with flatMap() in Parallel Streams

- The Java flatMap() implementation oddly forces sequential processing

```
<R> Stream<R> flatMap
   (Function<? super P_OUT,
    ? extends Stream<? extends R>>
    mapper) {
 ...
  public void accept(P_OUT u) {
    try(Stream<? extends R> result
        = mapper.apply(u)) {
    if (result != null) {
      if (...) {
        result
        .sequential()
        .forEach(downstream);
      ...
}
```

BEWARE!

This code always runs sequentially for "inner streams" that use flatMap()

See stackoverflow.com/questions/45038120/parallel-flatmap-always-sequential/66386078

Limitations with flatMap() in Parallel Streams

- This limitation renders flatMap() useless for parallel streams

```
var result = IntStream
  .rangeClosed(1, outerCount)
  .boxed()
  .parallel()

  .flatMap(innerCount -> IntStream
     .rangeClosed(1, innerCount)
     .boxed()
     .parallel())

  .anyMatch(...);
```

Limitations with flatMap() in Parallel Streams

- This limitation renders flatMap() useless for parallel streams

```
var result = IntStream
   .rangeClosed(1, outerCount)
   .boxed()
   .parallel()

   .flatMap(innerCount -> IntStream
      .rangeClosed(1, innerCount)
      .boxed()
      .parallel())

   .anyMatch(...);
```

The outer stream emits a parallel stream of Integer objects from 1 to outerCount

Limitations with flatMap() in Parallel Streams

- This limitation renders flatMap() useless for parallel streams

```
var result = IntStream
    .rangeClosed(1, outerCount)
    .boxed()
    .parallel()

    .flatMap(innerCount -> IntStream
        .rangeClosed(1, innerCount)
        .boxed()
        .parallel())

    .anyMatch(...);
```

Try using flatMap() to create an inner stream that emits Integer objects from 1 to innerCount in parallel

Limitations with flatMap() in Parallel Streams

- This limitation renders flatMap() useless for parallel streams

```
var result = IntStream
   .rangeClosed(1, outerCount)
   .boxed()
   .parallel()

   .flatMap(innerCount -> IntStream
      .rangeClosed(1, innerCount)
      .boxed()
      .parallel())

   .anyMatch(...);
```

Return true if all results are sequential, else false

Limitations with flatMap() in Parallel Streams

- This limitation renders flatMap() useless for parallel streams

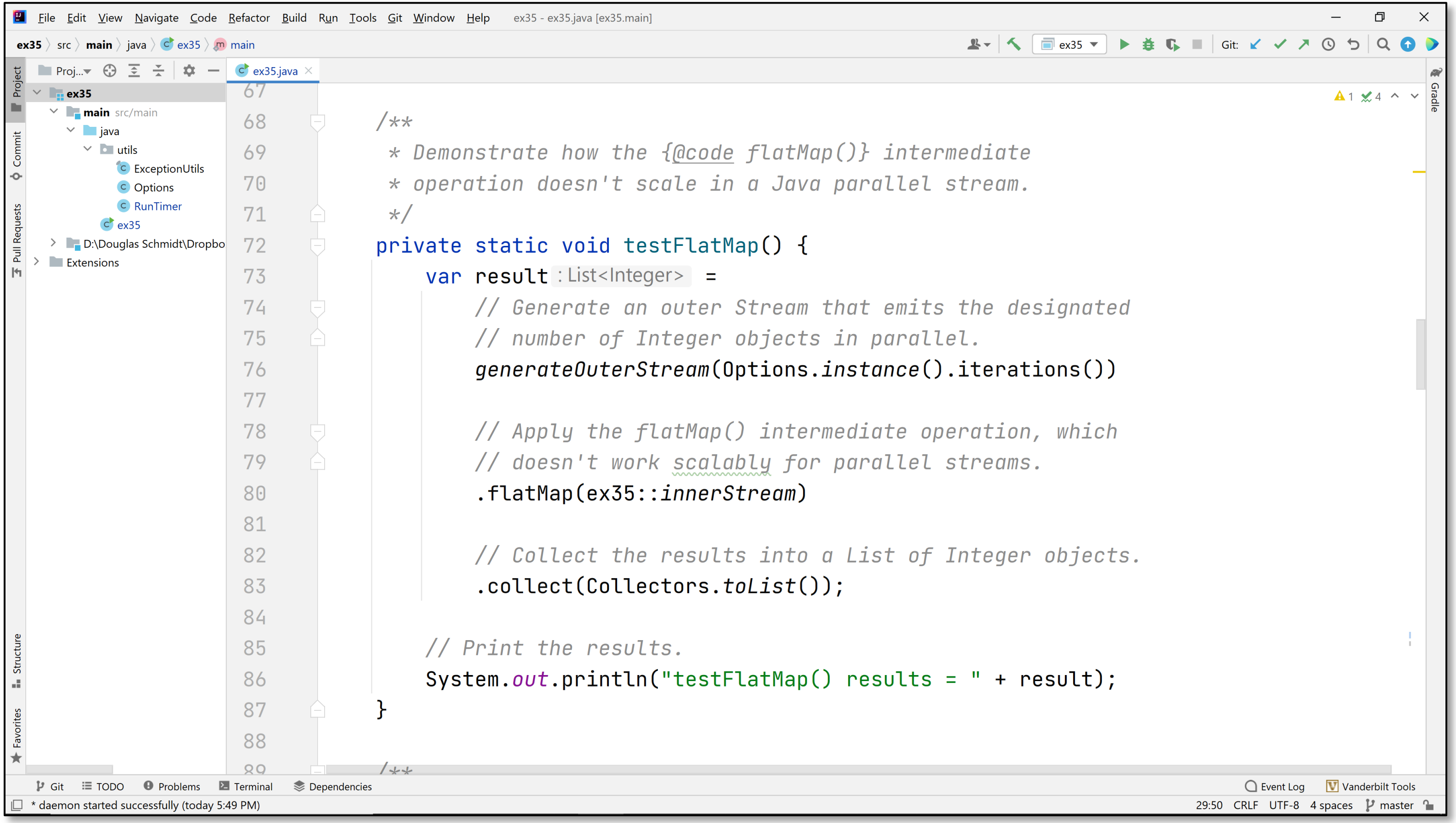
```
var result = IntStream
  .rangeClosed(1, outerCount)
  .boxed()
  .parallel()

  .flatMap(innerCount -> IntStream
     .rangeClosed(1, innerCount)
     .boxed()
     .parallel())

  .anyMatch(...);
```

This inner stream always runs sequentially for even though it designates .parallel() due to limitations with flatMap()

Limitations with flatMap() in Parallel Streams



File Edit View Navigate Code Refactor Build Run Tools Git Window Help ex35 - ex35.java [ex35.main]
ex35 src main java ex35 main
ex35.java
Project Commit Pull Requests Structure Favorites Gradle
ex35
main src/main
java
utils
ExceptionUtils
Options
RunTimer
ex35
D:\Douglas Schmidt\Dropbo
Extensions
Git TODO Problems Terminal Dependencies Event Log Vanderbilt Tools

```java
    /**
     * Demonstrate how the {@code flatMap()} intermediate
     * operation doesn't scale in a Java parallel stream.
     */
    private static void testFlatMap() {
        var result : List<Integer>  =
            // Generate an outer Stream that emits the designated
            // number of Integer objects in parallel.
            generateOuterStream(Options.instance().iterations())

            // Apply the flatMap() intermediate operation, which
            // doesn't work scalably for parallel streams.
            .flatMap(ex35::innerStream)

            // Collect the results into a List of Integer objects.
            .collect(Collectors.toList());

        // Print the results.
        System.out.println("testFlatMap() results = " + result);
    }

    /**
```

* daemon started successfully (today 5:49 PM)   29:50 CRLF UTF-8 4 spaces master

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex35

Overcoming Limitations with flatMap() in Parallel Streams

Overcoming Limitations with flatMap() in Parallel Streams

- One workaround is to use reduce() with Stream.concat()

```
var result = IntStream
    .rangeClosed(1, outerCount)
    .boxed()
    .parallel()

    .map(innerCount -> IntStream
        .rangeClosed(1, innerCount)
        .boxed()
        .parallel())

    .reduce(Stream::concat)
    .orElse(Stream.empty())

    .anyMatch(...);
```

Overcoming Limitations with flatMap() in Parallel Streams

- One workaround is to use reduce() with Stream.concat()

```
var result = IntStream
  .rangeClosed(1, outerCount)
  .boxed()
  .parallel()

  .map(innerCount -> IntStream
     .rangeClosed(1, innerCount)
     .boxed()
     .parallel())

  .reduce(Stream::concat)
  .orElse(Stream.empty())

  .anyMatch(...);
```

The outer stream emits a parallel stream of Integer objects from 1 to outerCount

Overcoming Limitations with flatMap() in Parallel Streams

- One workaround is to use reduce() with Stream.concat()

```
var result = IntStream
   .rangeClosed(1, outerCount)
   .boxed()
   .parallel()

   .map(innerCount -> IntStream
      .rangeClosed(1, innerCount)
      .boxed()
      .parallel())

   .reduce(Stream::concat)
   .orElse(Stream.empty())

   .anyMatch(...);
```

Use map() to create an inner stream-of-streams that emits Integer objects from 1 to innerCount in parallel

Overcoming Limitations with flatMap() in Parallel Streams

- One workaround is to use reduce() with Stream.concat()

```
var result = IntStream
  .rangeClosed(1, outerCount)
  .boxed()
  .parallel()

  .map(innerCount -> IntStream
     .rangeClosed(1, innerCount)
     .boxed()
     .parallel())

  .reduce(Stream::concat)
  .orElse(Stream.empty())

  .anyMatch(...);
```

Manually flatten the stream-of-streams into a stream of Integer objects

Overcoming Limitations with flatMap() in Parallel Streams

- One workaround is to use reduce() with Stream.concat()

```
var result = IntStream
   .rangeClosed(1, outerCount)
   .boxed()
   .parallel()

   .map(innerCount -> IntStream
      .rangeClosed(1, innerCount)
      .boxed()
      .parallel())

   .reduce(Stream::concat)
   .orElse(Stream.empty())

   .anyMatch(...);
```

Needed to handle the case where the stream is empty

Overcoming Limitations with flatMap() in Parallel Streams

- One workaround is to use reduce() with Stream.concat()

```
var result = IntStream
  .rangeClosed(1, outerCount)
  .boxed()
  .parallel()

  .map(innerCount -> IntStream
     .rangeClosed(1, innerCount)
     .boxed()
     .parallel())

  .reduce(Stream::concat)
  .orElse(Stream.empty())

  .anyMatch(...);
```

Return true if all results are sequential, else false

Overcoming Limitations with flatMap() in Parallel Streams

- One workaround is to use reduce() with Stream.concat()

```
var result = IntStream
   .rangeClosed(1, outerCount)
   .boxed()
   .parallel()

   .map(innerCount -> IntStream
      .rangeClosed(1, innerCount)
      .boxed()
      .parallel())

   .reduce(Stream::concat)
   .orElse(Stream.empty())

   .anyMatch(...);
```

This inner stream now runs in parallel, as intended

Overcoming Limitations with flatMap() in Parallel Streams

- Another workaround is to use mapMulti()

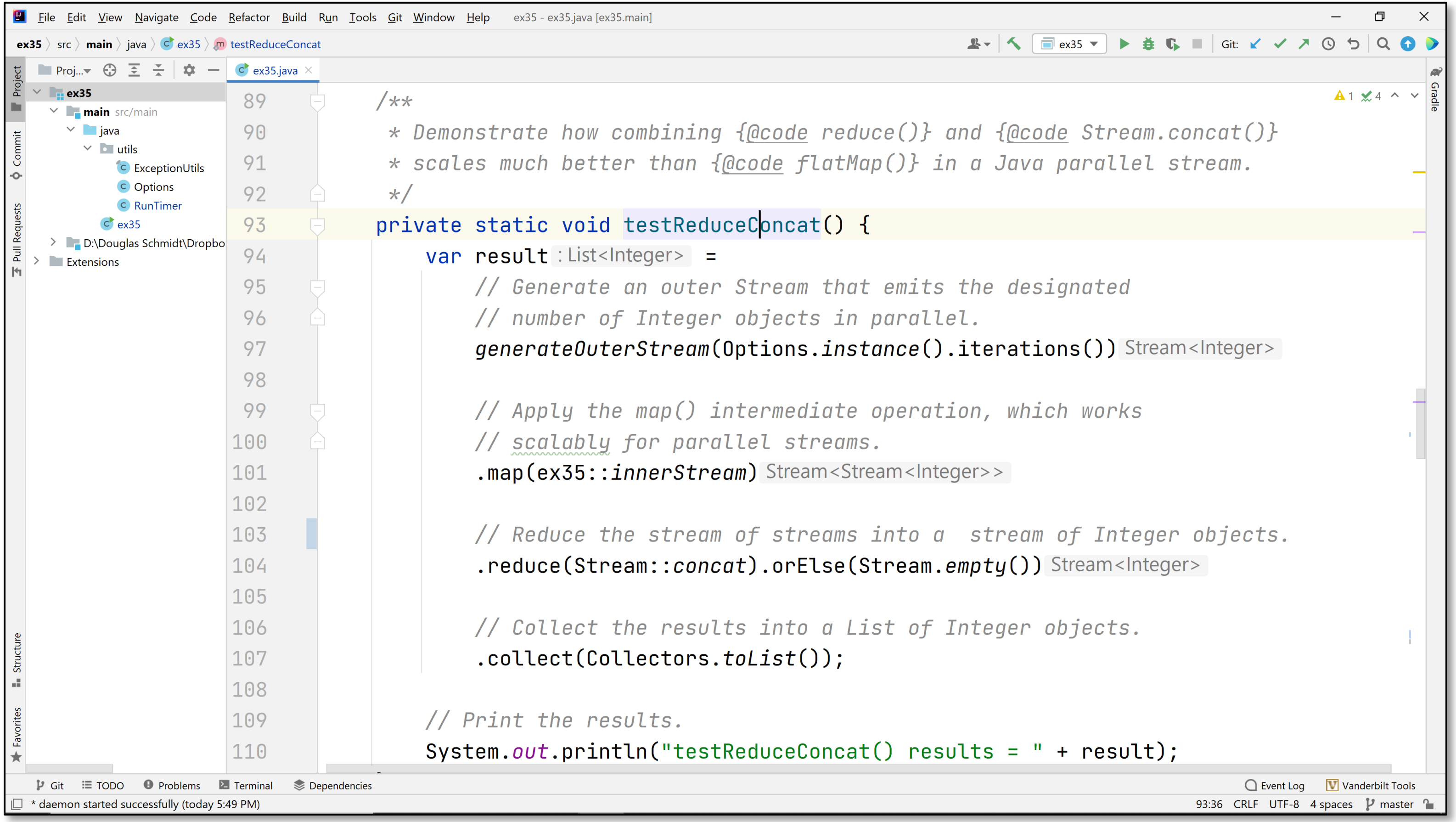
```
var result = IntStream
  .rangeClosed(1, outerCount)
  .boxed()
  .parallel()

  .mapMulti((innerCount,
             consumer) -> {
     int result = IntStream
       .rangeClosed(1, innerCount)
       .parallel()
       .mapMulti((i, c) -> ...)
       .sum();
     consumer.accept(result);})

  .allMatch(...);
```

This inner stream now also runs in parallel, as intended

Overcoming Limitations with flatMap() in Parallel Streams


File Edit View Navigate Code Refactor Build Run Tools Git Window Help
ex35 - ex35.java [ex35.main]
ex35 〉 src 〉 main 〉 java 〉 ex35 〉 testReduceConcat
ex35
Git:
Proj...
ex35.java
ex35
main src/main
java
utils
ExceptionUtils
Options
RunTimer
ex35
D:\Douglas Schmidt\Dropbo
Extensions
Project
Commit
Pull Requests
Structure
Favorites
Gradle
Git
TODO
Problems
Terminal
Dependencies
Event Log
Vanderbilt Tools
* daemon started successfully (today 5:49 PM)
93:36 CRLF UTF-8 4 spaces master

```java
    /**
     * Demonstrate how combining {@code reduce()} and {@code Stream.concat()}
     * scales much better than {@code flatMap()} in a Java parallel stream.
     */
    private static void testReduceConcat() {
        var result : List<Integer> =
            // Generate an outer Stream that emits the designated
            // number of Integer objects in parallel.
            generateOuterStream(Options.instance().iterations()) Stream<Integer>

            // Apply the map() intermediate operation, which works
            // scalably for parallel streams.
            .map(ex35::innerStream) Stream<Stream<Integer>>

            // Reduce the stream of streams into a  stream of Integer objects.
            .reduce(Stream::concat).orElse(Stream.empty()) Stream<Integer>

            // Collect the results into a List of Integer objects.
            .collect(Collectors.toList());

        // Print the results.
        System.out.println("testReduceConcat() results = " + result);
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex35

End of Java Parallel Streams Internals: Overcoming Limitations with flatMap() in Parallel Streams