Java Parallel Streams Internals:
Combining Results (Part 1)

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
- Perform a reduction to combine partial results into a single result
  - e.g., reduce() expects immutable objects, whereas collect() expects mutable result containers

See developer.ibm.com/languages/java/articles/j-java-streams-3-brian-goetz
Combining Results in a Parallel Stream
Combining Results in a Parallel Stream

- After the common fork-join pool finishes processing chunks their partial results are combined into a final result.

This discussion assumes a non-concurrent collector (other discussions follow).
Combining Results in a Parallel Stream

- After the common fork-join pool finishes processing chunks their partial results are combined into a final result
- `join()` occurs in a single thread at each level
  - i.e., the "parent"

Diagram:
- DataSource
  - DataSource₁
    - DataSource₁₁
    - DataSource₁₂
  - DataSource₂
    - DataSource₂₁
    - DataSource₂₂

Process sequentially:
- DataSource₁₁
- DataSource₁₂
- DataSource₂₁
- DataSource₂₂

"Children" join
- "Parent"
Combining Results in a Parallel Stream

- After the common fork-join pool finishes processing chunks their partial results are combined into a final result
- `join()` occurs in a single thread at each level
  - i.e., the “parent”

As a result, there’s typically no need for synchronizers during the joining
Combining Results in a Parallel Stream

• Different terminal operations combine partial results in different ways

Understanding these differences is particularly important for parallel streams
Combining Results in a Parallel Stream

- Different terminal operations combine partial results in different ways, e.g.
  - reduce() creates a new immutable value

See docs.oracle.com/javase/tutorial/essential/concurrency/immutable.html
Combining Results in a Parallel Stream

- Different terminal operations combine partial results in different ways, e.g.
  - reduce() creates a new immutable value

```java
long factorial(long n) {
    return LongStream
        .rangeClosed(1, n)
        .parallel()
        .reduce(1, (a, b) -> a * b,
                (a, b) -> a * b);
}
```

Combining Results in a Parallel Stream

Different terminal operations combine partial results in different ways, e.g.

- `reduce()` creates a new immutable value

```java
long factorial(long n) {
    return LongStream.of(longs 1..4)
        .parallel()
        .reduce(1, (a, b) -> a * b);
}
```

Combining Results in a Parallel Stream

• Different terminal operations combine partial results in different ways, e.g.
  • `reduce()` creates a new immutable value

```java
long factorial(long n) {
    return LongStream.rangeClosed(1, n)
        .parallel()
        .reduce(1, (a, b) -> a * b);
}
```

- `reduce()` combines two immutable values (e.g., `long`) & produces a new one

```
- `reduce()` creates a new immutable value
- `reduce()` combines two immutable values (e.g., `long`) & produces a new one
```

- Multiply pair-wise values

```
Range of longs from 1..8
  - longs 1..4
    - longs 1..2
      - Multiply pair-wise values
      - Process sequentially
      - 2
      - longs 3..4
        - Multiply pair-wise values
        - Process sequentially
        - 12
      - Process sequentially
      - 24
      - reduce()
      - 40,320
      - reduce()
      - 1,680
      - reduce()
      - 56
  - longs 5..8
    - longs 5..6
      - Multiply pair-wise values
      - Process sequentially
      - 30
      - Process sequentially
      - 1,680
      - 56
      - 30
      - 1,680
      - 56
      - reduce()
      - 1,680
      - reduce()
      - 56
```

- `reduce()` combines two immutable values (e.g., `long`) & produces a new one
Combining Results in a Parallel Stream

- Different terminal operations combine partial results in different ways, e.g.
  - `reduce()` creates a new immutable value
  - `collect()` mutates an existing value

See greenteapress.com/thinkapjava/html/thinkjava011.html
Combining Results in a Parallel Stream

- Different terminal operations combine partial results in different ways, e.g.
  - reduce() creates a new immutable value
  - collect() mutates an existing value

```java
Set<CharSequence> uniqueWords =
    getInput(sSHAKESPEARE, "\s+")
    .parallelStream()
    ...
    .collect(toCollection(TreeSet::new));
```

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex36](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex36)
Combining Results in a Parallel Stream

- Different terminal operations combine partial results in different ways, e.g.
  - `reduce()` creates a new immutable value
  - `collect()` mutates an existing value

```java
Set<CharSequence> uniqueWords =
    getInput(sSHAKESPEARE, "\s+")
    .parallelStream()
    ... .collect(toCollection(TreeSet::new));
```

`collect()` mutates a container to accumulate the result it's producing
Combining Results in a Parallel Stream

- Different terminal operations combine partial results in different ways, e.g.
  - `reduce()` creates a new immutable value
  - `collect()` mutates an existing value

```java
Set<CharSequence> uniqueWords =
    getInput(sSHAKESPEARE, "\s+")
    .parallelStream()
    ... .collect(ConcurrentSetCollector.toSet(ConcurrentHashMap::newKeySet));
```

Concurrent collectors (covered later) are different than non-concurrent collectors
End of Java Parallel Streams Internals: Combining Results (Part 1)