Understand Java Parallel Streams Internals: Combining Results (Part 1)

Douglas C. Schmidt

<u>d.schmidt@vanderbilt.edu</u>

www.dre.vanderbilt.edu/~schmidt



Professor of Computer Science

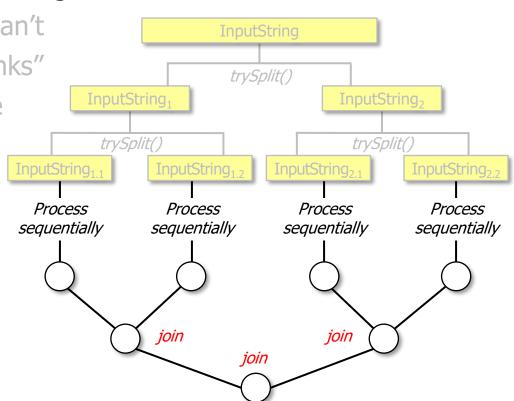
Institute for Software Integrated Systems

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



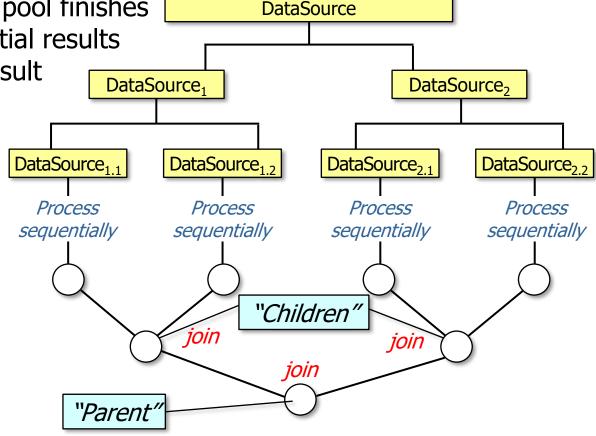
See <u>developer.ibm.com/languages/java/articles/j-java-streams-3-brian-goetz</u>

 After the common fork-join pool finishes **DataSource** processing chunks their partial results are combined into a final result DataSource₁ DataSource₂ DataSource_{1,2} DataSource_{2,2} DataSource_{2 1} DataSource_{1.1} **Process Process Process Process** sequentially sequentially sequentially sequentially **Partial** results join

This discussion assumes a non-concurrent collector (other discussions follow)

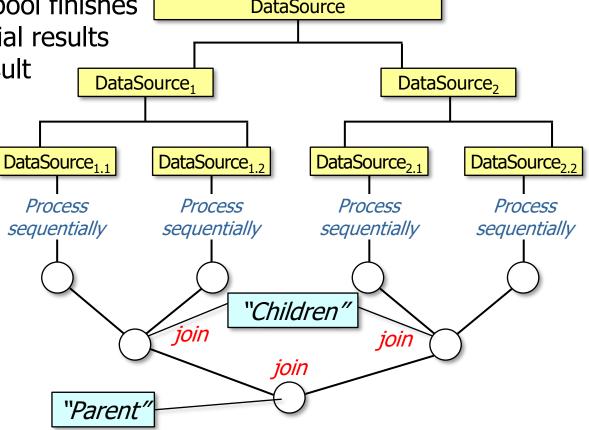
Final result

- 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"



- After the common fork-join pool finishes **DataSource**
- 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

 Different terminal operations combine partial results in different ways

Understanding these differences is particularly important for parallel streams

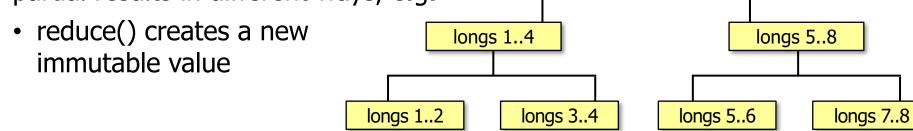
- Different terminal operations combine partial results in different ways, e.g.
 - reduce() creates a new immutable value



- Different terminal operations combine Range of longs from 1..8
 - partial results in different ways, e.g. reduce() creates a new

long factorial(long n) {

return LongStream



```
from 1..8 in parallel
.rangeClosed(1, n)
.parallel()
.reduce(1, (a, b) \rightarrow a * b,
          (a, b) -> a * b);
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex16

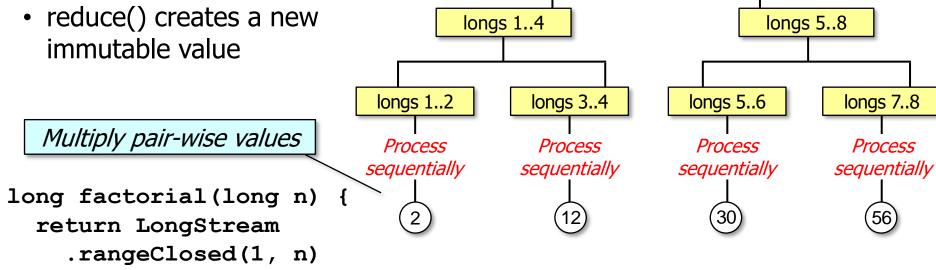
Generate a range of longs

- Different terminal operations combine Range of longs from 1..8
 - partial results in different ways, e.g.

.reduce(1, (a, b) -> a * b);

 reduce() creates a new immutable value

.parallel()



See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex16

 Different terminal operations combine Range of longs from 1..8 partial results in different ways, e.g. reduce() creates a new longs 5..8 longs 1..4 immutable value longs 5..6 longs 3..4 longs 7..8 longs 1..2 Multiply pair-wise values **Process Process Process Process** sequentially sequentially sequentially sequentially long factorial(long n) 56) return LongStream .rangeClosed(1, n) reduce() reduce() .parallel() 24) 1,680 reduce() .reduce(1, (a, b) -> a * b);

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

40,320

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



2nd quarter of words

Process

sequentially

2nd half of words

4th quarter of words

Process

sequentially

3rd quarter of words

Process

sequentially

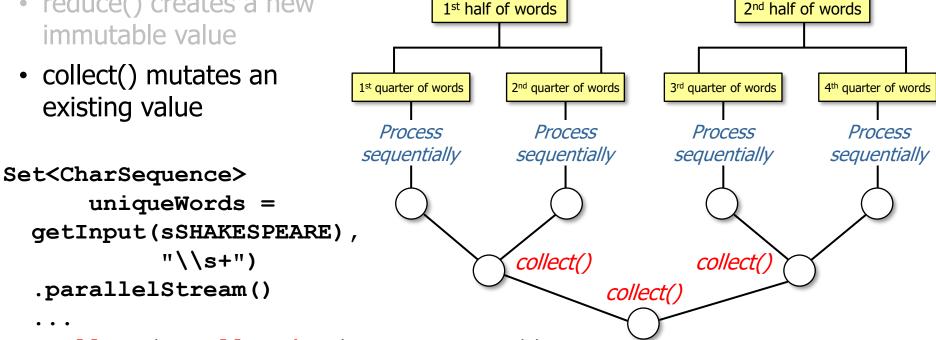
 Different terminal operations combine All words in Shakespeare's works

Process

- partial results in different ways, e.g. reduce() creates a new 1st half of words
 - immutable value
 - existing value
- collect() mutates an 1st quarter of words sequentially Set<CharSequence> uniqueWords = getInput(sSHAKESPEARE),
 - "\\s+")
 - .parallelStream()
 - .collect(toCollection(TreeSet::new));

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

- Different terminal operations combine All words in Shakespeare's works
- partial results in different ways, e.g. reduce() creates a new
 - existing value



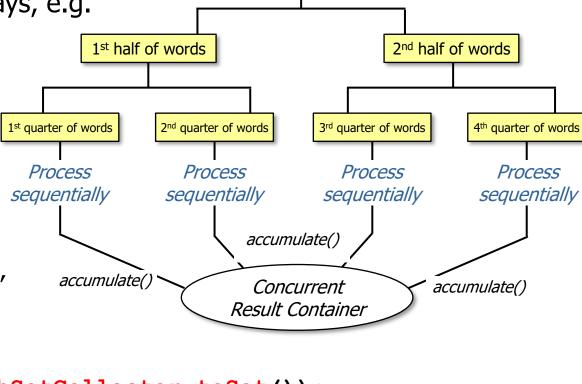
.collect(toCollection(TreeSet::new)); collect() mutates a container to accumulate the result it's producing

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

Set<CharSequence> uniqueWords = getInput(sSHAKESPEARE), "\\s+")

.parallelStream()

.collect(ConcurrentHashSetCollector.toSet()); Concurrent collectors (covered later) are different than non-concurrent collectors



End of Understand Java Parallel Streams Internals: Combining Results (Part 1)