Java Parallel Streams: Evaluating the Pros & Cons

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 Lesson

• Evaluate the pros & cons of Java parallel streams
Pros of Java
Parallel Streams
Pros of Java Parallel Streams

- The Java streams framework simplifies parallel programming by shielding developers from details of splitting, applying, & combining results.
• Parallel stream implementations are often (much) faster & more scalable than sequential (stream & loops) implementations
The performance speedup is largely a function of the partitioning strategy for the input (N), the amount of work performed (Q), & the # of cores.

The NQ model:
- N is the # of data elements to process per thread
- Q quantifies how CPU-intensive the processing is
Pros of Java Parallel Streams

- Apps often don’t need explicit synchronization or threading

Alleviates many accidental & inherent complexities of concurrency/parallelism
Pros of Java Parallel Streams

- Apps often don’t need explicit synchronization or threading
- Stateless behaviors alleviate the need to access shared mutable state

```java
return new SearchResults(
    Thread.currentThread().getId(),
    currentCycle(), phrase, title,
    StreamSupport
        .stream(new PhraseMatchSpliterator
            (input, phrase),
            parallel)
        .collect(toList()));
```

Return mLis.size() == 0;

See en.wikipedia.org/wiki/Pure_function
Pros of Java Parallel Streams

- Apps often don’t need explicit synchronization or threading
- Stateless behaviors alleviate the need to access shared mutable state
- Java class library handles locking needed to protect shared mutable state

See docs.oracle.com/javase/tutorial/essential/concurrency/collections.html
Pros of Java Parallel Streams

- Streams ensures that the structure of sequential & parallel code is the same

```java
List<List<SearchResults>> processStream() {
    return getInput().stream()
        .map(this::processInput)
        .collect(toList());
}

List<List<SearchResults>> processStream() {
    return getInput().parallelStream()
        .map(this::processInput)
        .collect(toList());
}
```

Converting sequential to parallel streams only require minuscule changes!
Pros of Java Parallel Streams

- Streams ensures that the structure of sequential & parallel code is the same

```java
List<SearchResults> results = mPhrasesToFind
    .parallelStream()
    .map(phase ->
        searchForPhrase(...,
            false))
    .filter(not(SearchResults::isEmpty))
    .collect(toList());
```

Converting sequential to parallel streams only require minuscule changes!
Pros of Java Parallel Streams

- Examples show synergies between functional & object-oriented programming

![Diagram showing intersections between procedural, object-oriented, functional, and logic programming paradigms.](image-url)

- Procedural: e.g., C, FORTRAN
- Object-Oriented: e.g., C#, Java, C++
- Functional: e.g., ML, Haskell
- Logic: e.g., Prolog
Pros of Java Parallel Streams

- Object-oriented design & programming features simplify understandability, reusability, & extensibility

Object-oriented techniques emphasize systematic reuse of structure
Pros of Java Parallel Streams

- Implementing object-oriented hook methods with functional programming features helps to close gap between domain intent & computations

```java
// Java Class

public class SearchWithParallelStreams {

    // Process the stream
    public List<List<SearchResults>> processStream() {
        return mPhrasesToFind.parallelStream()
                .map(phrase -> searchForPhrase(phrase, input, title, false))
                .filter(not(SearchResults::isEmpty))
                .collect(toList());
    }

    // Get input
    public String getInput() {
        return parallelStream().map(this::processInput).collect(toList());
    }
}
```
Cons of Java
Parallel Streams
Cons of Java Parallel Streams

- There are some limitations with Java parallel streams

The Java parallel streams framework is not all unicorns & rainbows!!
Cons of Java Parallel Streams

- There are some limitations with Java parallel streams, e.g.
- Some problems can’t be expressed via the “split-apply-combine” model

See dzone.com/articles/whats-wrong-java-8-part-iii
Cons of Java Parallel Streams

• There are some limitations with Java parallel streams, e.g.
  • Some problems can’t be expressed via the “split-apply-combine” model
  • If behaviors aren’t stateless & thread-safe race conditions may occur

Race conditions occur when a program depends on the sequence or timing of threads for it to operate properly

See en.wikipedia.org/wiki/Race_condition#Software
Cons of Java Parallel Streams

- There are some limitations with Java parallel streams, e.g.
  - Some problems can’t be expressed via the “split-apply-combine” model
  - If behaviors aren’t stateless & thread-safe race conditions may occur
  - Parallel spliterators may be tricky...

See lesson on “Java SearchWithParallelSpliterator Example: trySplit()”
Cons of Java Parallel Streams

- There are some limitations with Java parallel streams, e.g.
  - Some problems can’t be expressed via the “split-apply-combine” model
  - If behaviors aren’t stateless & thread-safe race conditions may occur
  - Parallel spliterators may be tricky…
    - Concurrent collectors are easier

See lesson on “Java Parallel Stream Internals: Non-Concurrent & Concurrent Collectors”
Cons of Java Parallel Streams

- There are some limitations with Java parallel streams, e.g.
  - Some problems can’t be expressed via the “split-apply-combine” model
  - If behaviors aren’t stateless & thread-safe race conditions may occur
  - Parallel spliterators may be tricky...
  - All parallel streams share a common fork-join pool

See dzone.com/articles/think-twice-using-java-8
Cons of Java Parallel Streams

- There are some limitations with Java parallel streams, e.g.
  - Some problems can’t be expressed via the “split-apply-combine” model
  - If behaviors aren’t stateless & thread-safe race conditions may occur
  - Parallel spliterators may be tricky...
  - All parallel streams share a common fork-join pool
    - Java completable futures don’t have this limitation

See dzone.com/articles/think-twice-using-java-8
Cons of Java Parallel Streams

- There are some limitations with Java parallel streams, e.g.
  - Some problems can’t be expressed via the “split-apply-combine” model
  - If behaviors aren’t stateless & thread-safe race conditions may occur
  - Parallel spliterators may be tricky...
  - All parallel streams share a common fork-join pool
    - Java completable futures don’t have this limitation
    - It’s important to know how to apply ManagedBlockers

See “The Java Fork-Join Pool: Applying the ManagedBlocker Interface”
Cons of Java Parallel Streams

• There are some limitations with Java parallel streams, e.g.
  • Some problems can’t be expressed via the “split-apply-combine” model
  • If behaviors aren’t stateless & thread-safe race conditions may occur
  • Parallel spliterators may be tricky...
  • All parallel streams share a common fork-join pool
  • Streams incur some overhead
Cons of Java Parallel Streams

- There are some limitations with Java parallel streams, e.g.
  - Some problems can’t be expressed via the “split-apply-combine” model
  - If behaviors aren’t stateless & thread-safe race conditions may occur
  - Parallel spliterators may be tricky...
  - All parallel streams share a common fork-join pool
  - Streams incur some overhead, e.g.
    - Splitting & combining overhead

See blog.jooq.org/2015/12/08/3-reasons-why-you-shouldnt-replace-your-for-loops-by-stream-foreach
There are some limitations with Java parallel streams, e.g.

- Some problems can’t be expressed via the “split-apply-combine” model
- If behaviors aren’t stateless & thread-safe race conditions may occur
- Parallel spliterators may be tricky...
- All parallel streams share a common fork-join pool
- Streams incur some overhead, e.g.
  - Splitting & combining overhead
  - Fork-join framework

See coopsoft.com/dl/Blunder.pdf
Cons of Java Parallel Streams

- There are some limitations with Java parallel streams, e.g.
  - Some problems can’t be expressed via the “split-apply-combine” model
  - If behaviors aren’t stateless & thread-safe race conditions may occur
  - Parallel spliterators may be tricky...
  - All parallel streams share a common fork-join pool
  - Streams incur some overhead, e.g.
    - Splitting & combining overhead
    - Fork-join framework
  - Java completable futures may be more efficient & scalable
Cons of Java Parallel Streams

- There are some limitations with Java parallel streams, e.g.
  - Some problems can’t be expressed via the “split-apply-combine” model
  - If behaviors aren’t stateless & thread-safe race conditions may occur
  - Parallel spliterators may be tricky...
  - All parallel streams share a common fork-join pool
- Streams incur some overhead, e.g.
  - Splitting & combining overhead
  - Fork-join framework
  - Java completable futures may be more efficient & scalable

Naturally, your mileage may vary..
There are some limitations with Java parallel streams, e.g.

- Some problems can’t be expressed via the “split-apply-combine” model.
- If behaviors aren’t stateless & thread-safe race conditions may occur.
- Parallel spliterators may be tricky...
- All parallel streams share a common fork-join pool.
- Streams incur some overhead.
- There’s no substitute for benchmarking!

See [java-performance.info/jmh](http://java-performance.info/jmh)
Wrapping Up Java
Parallel Streams
In general, there's a tradeoff between computing performance & programmer productivity when choosing amongst these frameworks.

i.e., completable futures are more efficient & scalable, but are harder to program.
Wrapping Up Java Parallel Streams

• In general, however, the pros of Java parallel streams far outweigh the cons for many use cases!!

See www.ibm.com/developerworks/library/j-jvmc2
Wrapping Up Java Parallel Streams

- Good coverage of parallel streams appears in the book “Modern Java in Action”

End of Java Parallel Streams: Evaluating the Pros & Cons