Evaluate the Java SearchWith ParallelStreams Case Study

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

- Know how Java parallel streams are applied in `SearchWithParallelStreams`
- Understand the pros & cons of the `SearchWithParallelStreams` class

See `SearchStreamGang/src/main/java/livelessons/streamgangs/SearchWithParallelStreams.java`
Pros of the SearchWith ParallelStreams Class
Pros of the SearchWithParallelStreams Class

- This example shows that the difference between sequential & parallel streams is often minuscule!

See docs.oracle.com/javase/tutorial/collections/streams/parallelism.html
This example shows that the difference between sequential & parallel streams is often minuscule!

Here's `processStream()` from `SearchWithSequentialStreams` that we examined earlier.

```java
List<List<SearchResults>>
processStream() {
    return getInput()
        .stream()
        .map(this::processInput)
        .collect(toList());
}
```
This example shows that the difference between sequential & parallel streams is often minuscule!

Here’s `processStream()` in `SearchWithParallelStreams`:

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

VS

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

- This example shows that the difference between sequential & parallel streams is often minuscule!

\[
\text{List<List<SearchResults>\>}
\]

processStream() {
    return getInput()
    .stream()
    .map(this::processInput)
    .collect(toList());
}

VS

\[
\text{List<List<SearchResults>\>}
\]

processStream() {
    return getInput()
    .parallelStream()
    .map(this::processInput)
    .collect(toList());
}

Changing all the stream() calls to parallelStream() calls is the minuscule difference between implementations!!
Pros of the SearchWithParallelStreams Class

• This example shows that the difference between sequential & parallel streams is often minuscule!

• Moreover, substantial speedups can occur on multi-core processors!

Tests conducted on a 2.7GHz quad-core Lenovo P50 with 32 Gbytes of RAM
This example shows that the difference between sequential & parallel streams is often minuscule!

Moreover, substantial speedups can occur on multi-core processors!

Tests conducted on a 2.9GHz quad-core MacBook Pro with 16 Gbytes of RAM
This example shows that the difference between sequential & parallel streams is often minuscule!

Moreover, substantial speedups can occur on multi-core processors!

Superlinear speed-ups arise from “hyper-threaded” (virtual) cores

See en.wikipedia.org/wiki/Hyper-threading
Pros of the SearchWithParallelStreams Class

- This example shows that the difference between sequential & parallel streams is often minuscule!
- Moreover, substantial speedups can occur on multi-core processors!
- Superlinear speed-ups arise from “hyper-threaded” (virtual) cores
- Increases the # of independent instructions in the pipeline via a superscalar architecture

A superscalar processor can execute more than one instruction during a clock cycle by simultaneously dispatching multiple instructions to different execution units

See en.wikipedia.org/wiki/Superscalar_processor
Cons of the SearchWith ParallelStreams Class
Cons of the SearchWithParallelStreams Class

- Just because two minuscule changes are needed doesn’t mean this is the best implementation!

Other Java concurrency/parallelism strategies are even more efficient...

Tests conducted on a 2.7GHz quad-core Lenovo P50 with 32 Gbytes of RAM
Just because two minuscule changes are needed doesn’t mean this is the best implementation!

There’s no substitute for systematic benchmarking & experimentation
• We’ll show how to overcome these cons in an upcoming lesson that focuses on the SearchWithParallelSpliterator class.

```java
<<Java Class>>
SearchWithParallelSpliterator

- processStream(): List<List<SearchResults>>
- processInput(CharArraySequence): List<SearchResults>
```

Starting SearchStreamGangTest

PARALLEL_SPLITTERATOR executed in 409 msecs
COMPLETABLE_FUTURES_INPUTS executed in 426 msecs
COMPLETABLE_FUTURES_PHASES executed in 427 msecs
PARALLEL_STREAMS executed in 437 msecs
PARALLEL_STREAM_PHASES executed in 440 msecs
RXJAVA_PHASES executed in 485 msecs
PARALLEL_STREAM_INPUTS executed in 802 msecs
RXJAVA_INPUTS executed in 866 msecs
SEQUENTIAL_LOOPS executed in 1638 msecs
SEQUENTIAL_STREAM executed in 1958 msecs

Ending SearchStreamGangTest

See Searlivelessons/streamgangs/SearchWithParallelSpliterator.java
Cons of the SearchWithParallelStreams Class

- We’ll show how to overcome these cons in an upcoming lesson that focuses on the SearchWithParallelSpliterator class

SearchWithParallelSpliterator is thus the most aggressive parallelism strategy!
End of Evaluate the Java Search WithParallelStreams Case Study