Evaluating the Java Search With 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

<<Java Class>>

`SearchWithParallelStreams`

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

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!

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

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

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

Changing all the stream() calls to parallelStream() calls is the minuscule difference between implementations!!

Pros of the SearchWithParallelStreams Class

```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!

- 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
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

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

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

There’s no substitute for systematic benchmarking & experimentation.
Cons of the SearchWithParallelStreams Class

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

```
<<Java Class>>
SearchWithParallelSpliterator

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

Input Strings to Search

Search Phrases

See Searlivelessons/streamgangs/SearchWithParallelSpliterator.java
Cons of the `SearchWithParallelSpliterator` 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 Evaluating the Java Search WithParallelStreams Case Study