Java 8 SearchWithParallelStreams Example (Part 2)

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 8 parallel streams are applied in SearchWithParallelStreams

• Understand the pros & cons of the SearchWithParallelStreams class

<<Java Class>>

SearchWithParallelStreams

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

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

Here's `processStream()` from `SearchWithSequentialStream` that we examined earlier.
Pros of the SearchWithParallelStreams Class

- 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());
}
VS

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

Here’s `processStream()` in `SearchWithParallelStreams`
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
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 2.9GHz 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
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

### Superscalar Processor

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](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 8 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!

EPA Fuel Economy Estimates

These estimates reflect new EPA methods beginning with 2008 models:

CITY MPG 18
HIGHWAY MPG 25

Estimated Annual Fuel Cost $2,039
based on 15,000 miles at $2.80 per gallon

Combined Fuel Economy
This Vehicle
21
10 21 to 29 MPG
31

Your actual mileage will vary depending on how you drive and maintain your vehicle.

See the FREE Fuel Economy Guide at dealers or www.fueleconomy.gov

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.
End of Java 8 SearchWithParallelStreams Example (Part 2)