Java SearchWithParallelStreams Example: Evaluating Pros & Cons

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

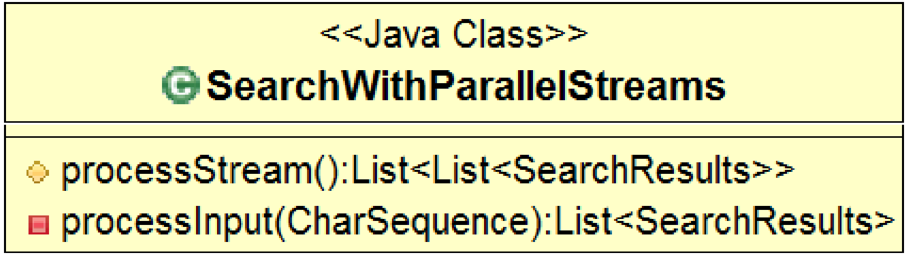
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this 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>>

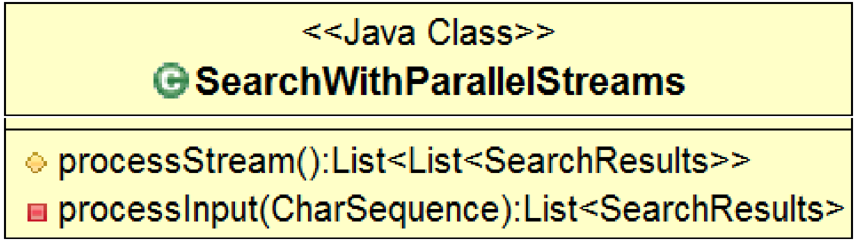
processInput(CharSequence):List<SearchResults>

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!

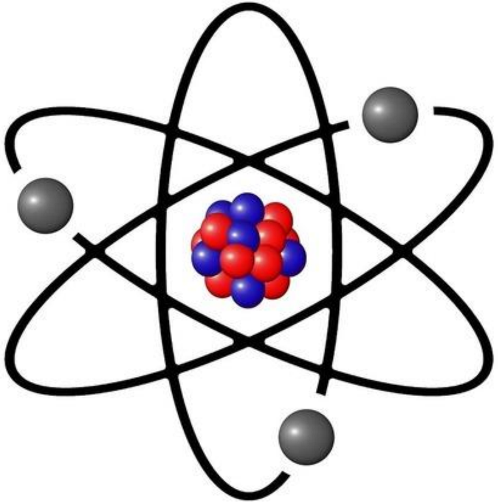


<<Java Class>>

SearchWithParallelStreams

◆ processStream():List<List<SearchResults>>

■ processInput(CharSequence):List<SearchResults>



See docs.oracle.com/javase/tutorial/collections/streams/parallelism.html

Pros of the SearchWithParallelStreams Class

- 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!

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

vs

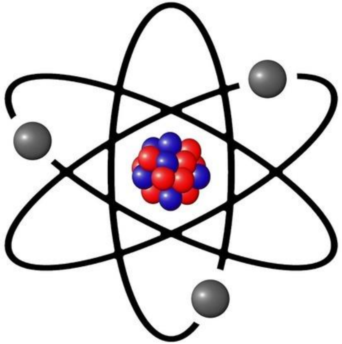
```
List<List<SearchResults>>
            processStream() {
  return getInput()
    .parallelStream()
    .map(this::processInput)
    .collect(toList());
}
```

Here's processStream() in SearchWithParallelStreams

Pros of the SearchWithParallelStreams Class

- 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!!



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

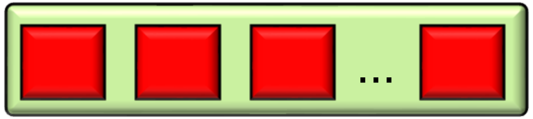
vs

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

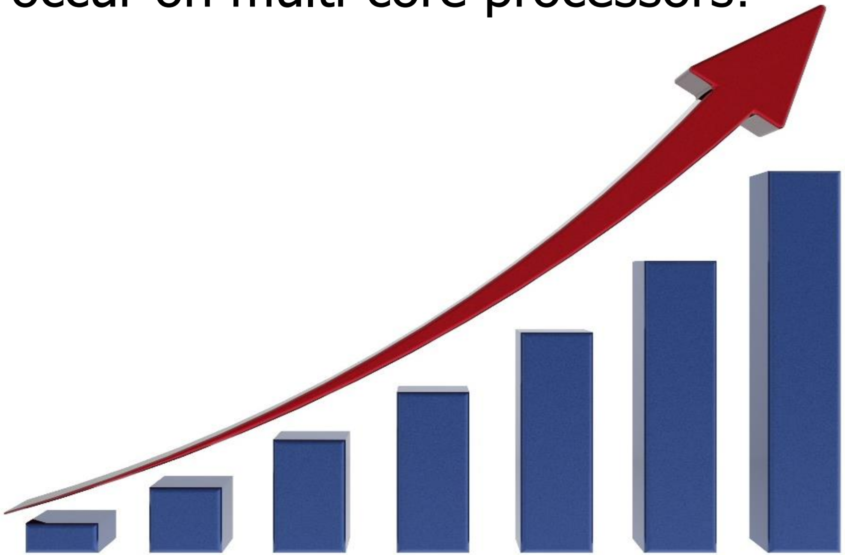
Pros of the SearchWithParallelStreams Class

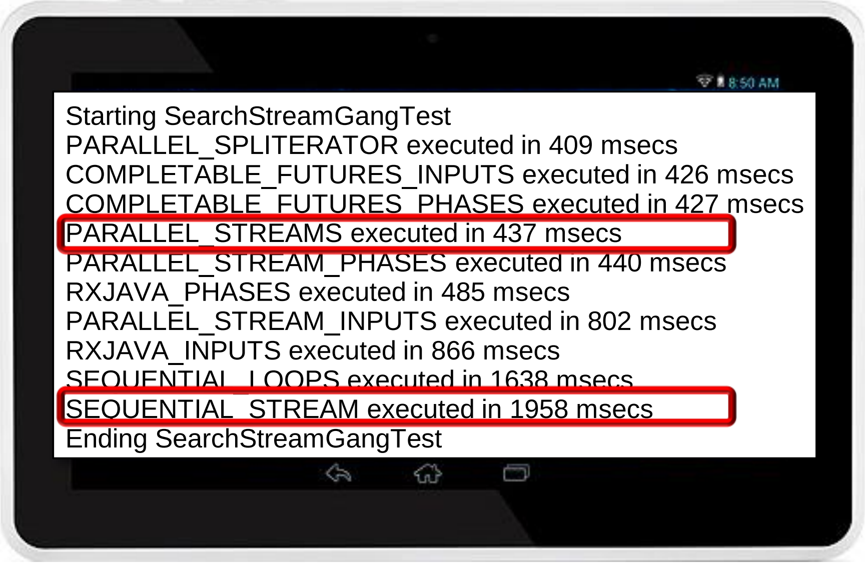
- This example shows that the difference between sequential & parallel streams is often minuscule!
  - Moreover, substantial speedups can occur on multi-core processors!

Input Strings to Search



Search Phrases



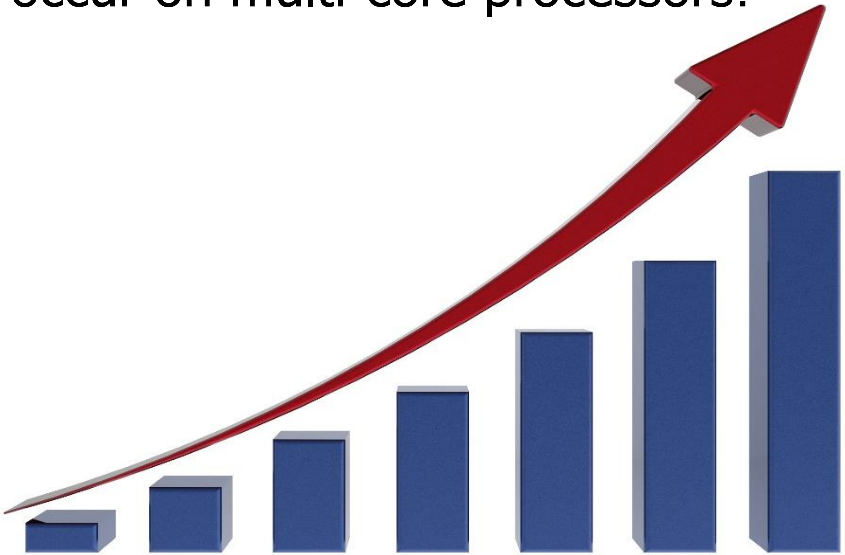


```
Starting SearchStreamGangTest
PARALLEL_SPLITERATOR executed in 409 msecs
COMPLETABLE_FUTURES_INPUTS executed in 426 msecs
COMPLETABLE_FUTURES_PHASES executed in 427 msecs
PARALLEL_STREAMS executed in 437 msecs
PARALLEL_STREAM_PHASES executed in 440 msecs
RXJAVA_PHASES executed in 485 msecs
PARALLEL_STREAM_INPUTS executed in 802 msecs
RXJAVA_INPUTS executed in 866 msecs
SEQUENTIAL_LOOPS executed in 1638 msecs
SEQUENTIAL_STREAM executed in 1958 msecs
Ending SearchStreamGangTest
```

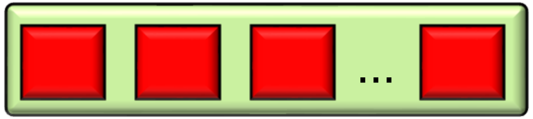
Tests conducted on a 2.7GHz quad-core Lenovo P50 with 32 Gbytes of RAM

Pros of the SearchWithParallelStreams Class

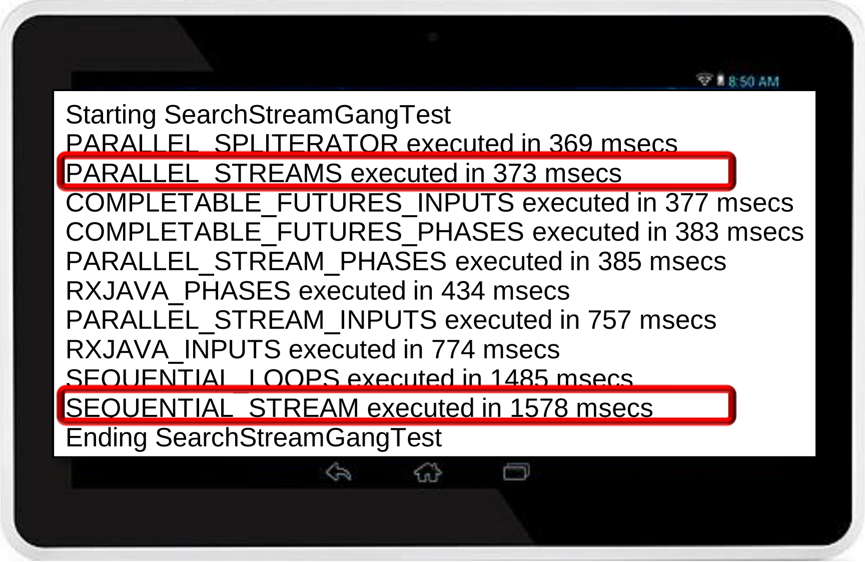
- This example shows that the difference between sequential & parallel streams is often minuscule!
  - Moreover, substantial speedups can occur on multi-core processors!



Input Strings to Search



Search Phrases

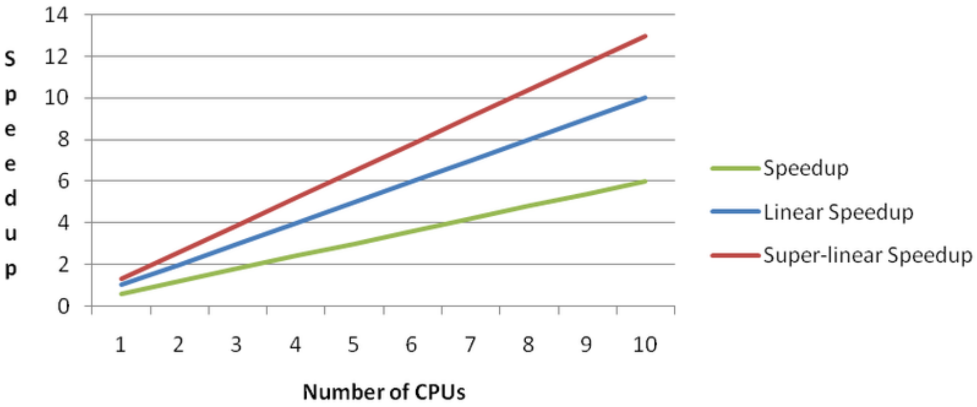


```
Starting SearchStreamGangTest
PARALLEL_SPLITERATOR executed in 369 msecs
PARALLEL_STREAMS executed in 373 msecs
COMPLETABLE_FUTURES_INPUTS executed in 377 msecs
COMPLETABLE_FUTURES_PHASES executed in 383 msecs
PARALLEL_STREAM_PHASES executed in 385 msecs
RXJAVA_PHASES executed in 434 msecs
PARALLEL_STREAM_INPUTS executed in 757 msecs
RXJAVA_INPUTS executed in 774 msecs
SEQUENTIAL_LOOPS executed in 1485 msecs
SEQUENTIAL_STREAM executed in 1578 msecs
Ending SearchStreamGangTest
```

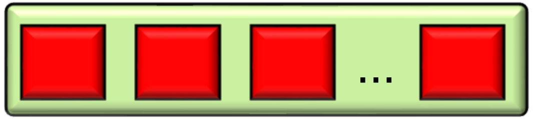
Tests conducted on a 2.9GHz quad-core MacBook Pro with 16 Gbytes of RAM

Pros of the SearchWithParallelStreams Class

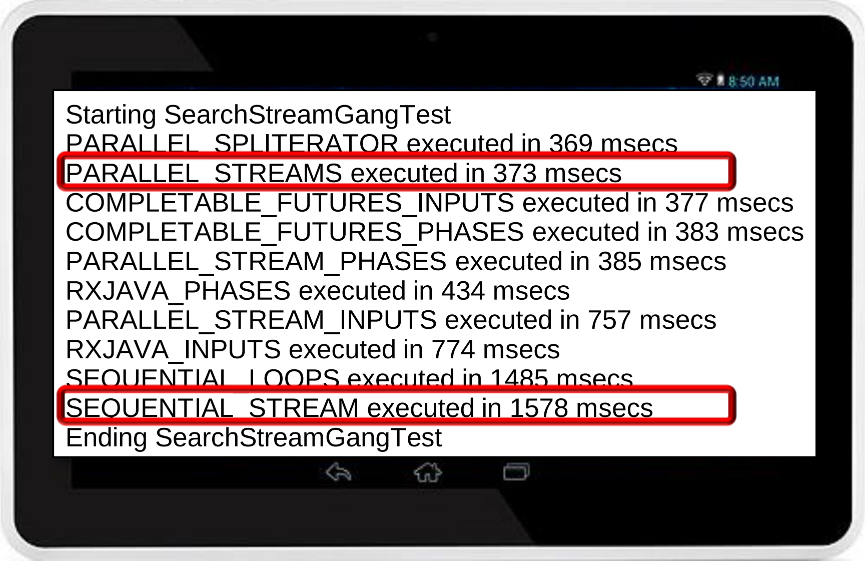
- This example shows that the difference between sequential & parallel streams is often minuscule!
  - Moreover, substantial speedups can occur on multi-core processors!
  - Superlinear speed-ups arise from "hyper-threaded" (virtual) cores



Input Strings to Search



Search Phrases

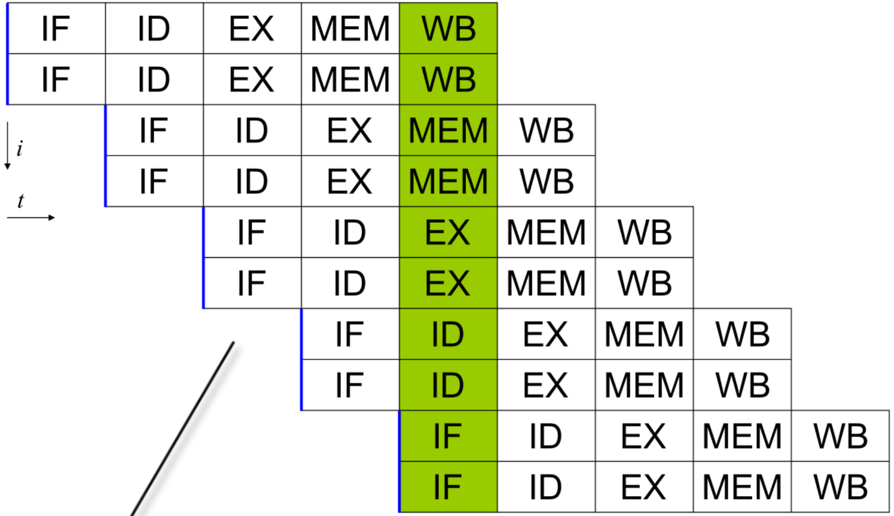


```
Starting SearchStreamGangTest
PARALLEL_SPLITERATOR executed in 369 msecs
PARALLEL_STREAMS executed in 373 msecs
COMPLETABLE_FUTURES_INPUTS executed in 377 msecs
COMPLETABLE_FUTURES_PHASES executed in 383 msecs
PARALLEL_STREAM_PHASES executed in 385 msecs
RXJAVA_PHASES executed in 434 msecs
PARALLEL_STREAM_INPUTS executed in 757 msecs
RXJAVA_INPUTS executed in 774 msecs
SEQUENTIAL_LOOPS executed in 1485 msecs
SEQUENTIAL_STREAM executed in 1578 msecs
Ending SearchStreamGangTest
```

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



IF	ID	EX	MEM	WB				
IF	ID	EX	MEM	WB				
	IF	ID	EX	MEM	WB			
	IF	ID	EX	MEM	WB			
		IF	ID	EX	MEM	WB		
		IF	ID	EX	MEM	WB		
			IF	ID	EX	MEM	WB	
			IF	ID	EX	MEM	WB	
				IF	ID	EX	MEM	WB
				IF	ID	EX	MEM	WB

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

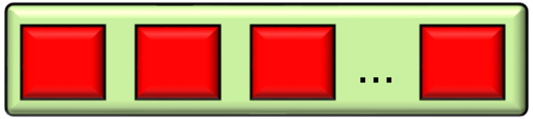
Cons of the SearchWith ParallelStreams Class

Cons of the SearchWithParallelStreams Class

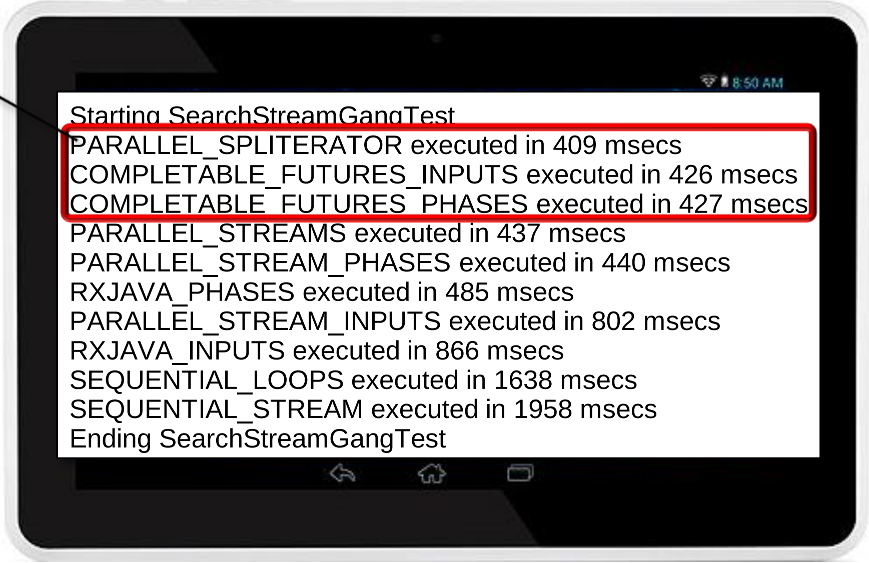
- Just because two minuscule changes are needed doesn't mean this is the best implementation!

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

Input Strings to Search



Search Phrases

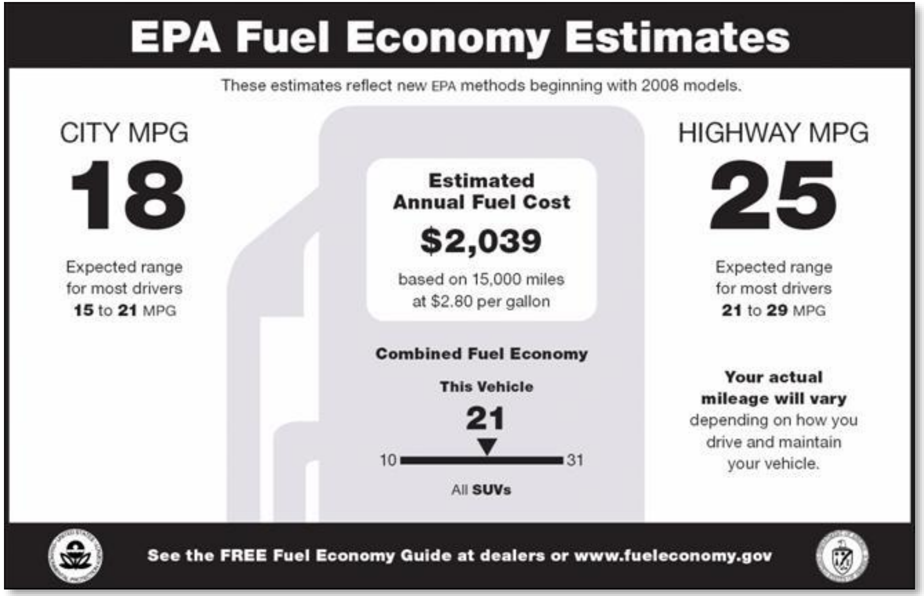


```
Starting SearchStreamGangTest
PARALLEL_SPLITERATOR executed in 409 msecs
COMPLETABLE_FUTURES_INPUTS executed in 426 msecs
COMPLETABLE_FUTURES_PHASES executed in 427 msecs
PARALLEL_STREAMS executed in 437 msecs
PARALLEL_STREAM_PHASES executed in 440 msecs
RXJAVA_PHASES executed in 485 msecs
PARALLEL_STREAM_INPUTS executed in 802 msecs
RXJAVA_INPUTS executed in 866 msecs
SEQUENTIAL_LOOPS executed in 1638 msecs
SEQUENTIAL_STREAM executed in 1958 msecs
Ending SearchStreamGangTest
```

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
Expected range for most drivers
15 to 21 MPG

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

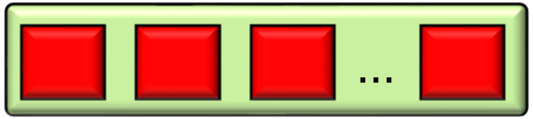
Combined Fuel Economy
This Vehicle
21
10 31
All SUVs

HIGHWAY MPG
25
Expected range for most drivers
21 to 29 MPG

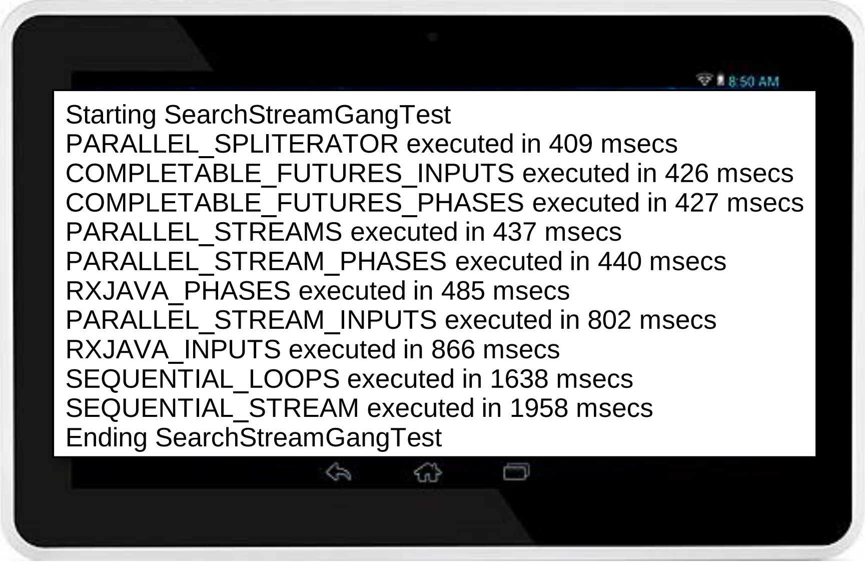
Your actual mileage will vary depending on how you drive and maintain your vehicle.

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

Input Strings to Search



Search Phrases

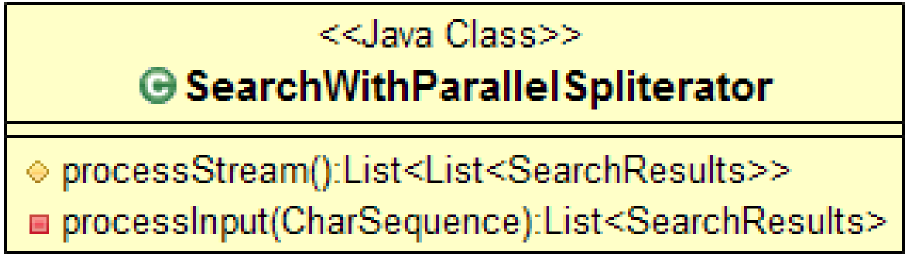


```
Starting SearchStreamGangTest
PARALLEL_SPLITERATOR executed in 409 msecs
COMPLETABLE_FUTURES_INPUTS executed in 426 msecs
COMPLETABLE_FUTURES_PHASES executed in 427 msecs
PARALLEL_STREAMS executed in 437 msecs
PARALLEL_STREAM_PHASES executed in 440 msecs
RXJAVA_PHASES executed in 485 msecs
PARALLEL_STREAM_INPUTS executed in 802 msecs
RXJAVA_INPUTS executed in 866 msecs
SEQUENTIAL_LOOPS executed in 1638 msecs
SEQUENTIAL_STREAM executed in 1958 msecs
Ending SearchStreamGangTest
```

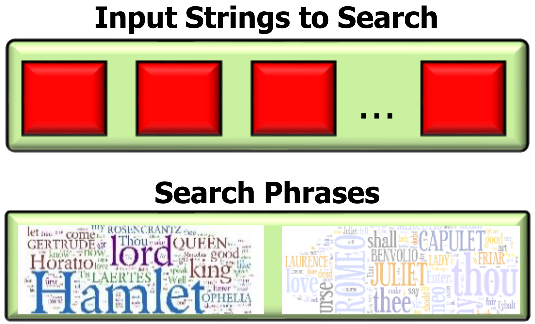
There's no substitute for systematic benchmarking & experimentation

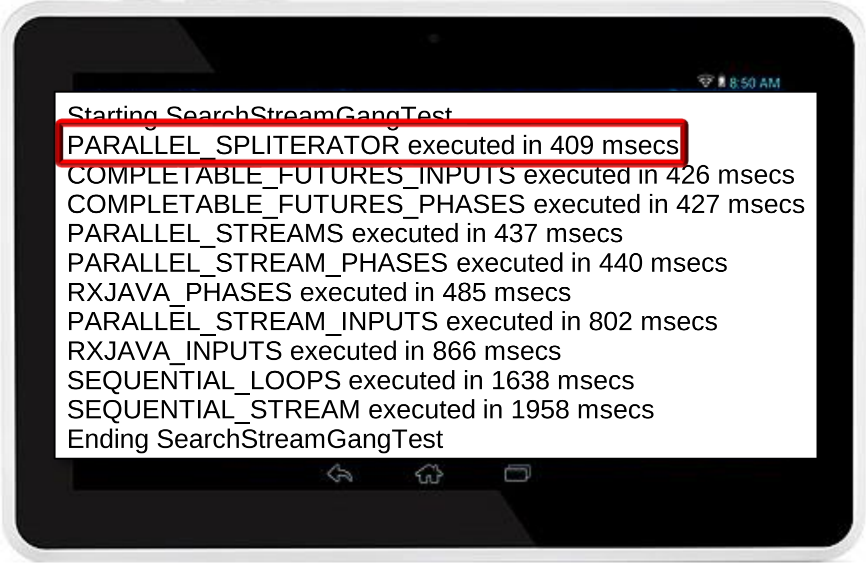
Cons of the SearchWithParallelStreams Class

- 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<SearchResults>




```
Starting SearchStreamGangTest
PARALLEL_SPLITERATOR 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 SearchStreamGang/src/main/java/livelessons/streamgangs/SearchWithParallelSpliterator.java

Cons of the SearchWithParallelStreams Class

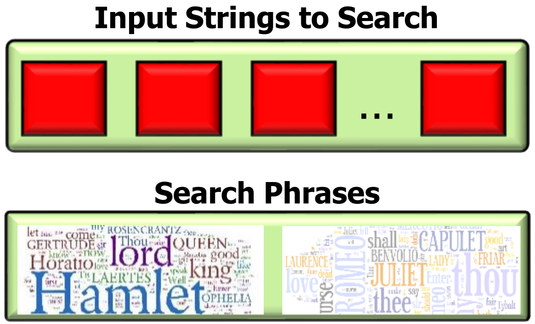
- We'll show how to overcome these cons in an upcoming lesson that focuses on the SearchWithParallelSpliterator class

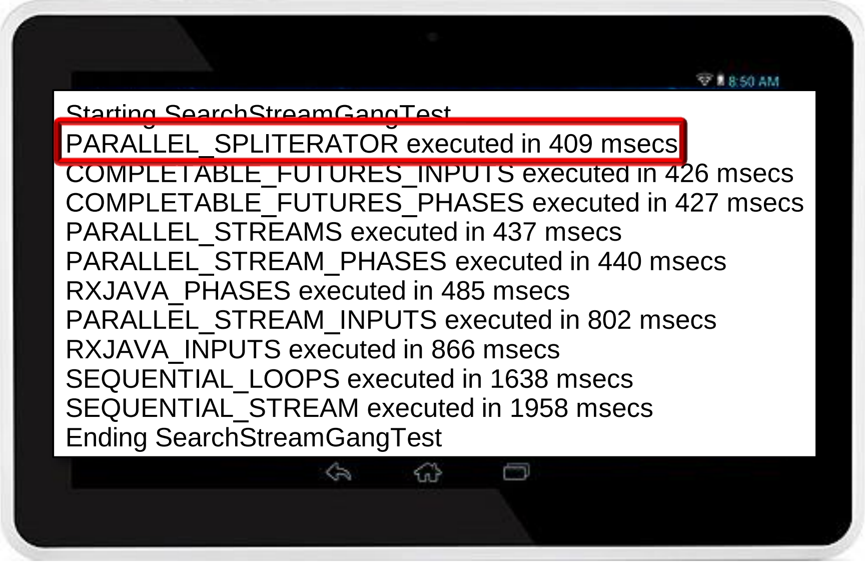
<<Java Class>>

SearchWithParallelSpliterator

process ts>>

process hResults>





```
Starting SearchStreamGangTest
PARALLEL_SPLITERATOR 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
```

SearchWithParallelSpliterator is thus the most aggressively parallelism strategy!

End of Java SearchWith ParallelStreams Example: Evaluating Pros & Cons