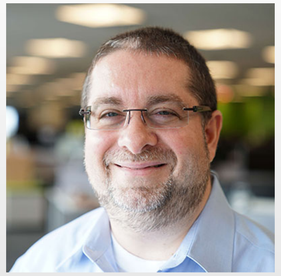
JVMs.Compare

Benchmark Performance of Different JDK/JVM combos

Chandra Guntur & Donald Raab

About Us

- Java Champions
- JCP Executive Committee Reps. for BNY Mellon
- Ardent bloggers and tweeters



- Director at BNY Mellon
- Working in Financial Tech. since 2003
- Programming in Java since 1998
- JUG Leader @ NYJavaSIG
- Creator of Java-Katas Github repository

- Managing Director at BNY Mellon
- 18+ years of experience in Financial Tech.
- Programming in Java since 1997
- Member of JSR 335 Expert Group
- Creator of Eclipse Collection Java Library

Session Agenda

Compare benchmarks for operations using:

Eclipse Collections and JDK Java Collection Framework:

- on a primitive IntList & List<Integer>
- on a List<Person>

using several JDK/JVM combinations

Session Agenda

Compare benchmarks for operations using:

Eclipse Collections and JDK Java Collection Framework:

- on a primitive IntList & List<Integer>
- on a List<Person>

using several JDK/JVM combinations

- No tweaks to VM Options or Flags
- Intent to test out-of-the-box throughputs

WHY?

- Show Java developers what the comparisons reveal.
- Show JVM developers what can be improved if there is a target audience for some metric that is not currently high in throughput.

JMH Disclaimers

JMH is tuned for OpenJDK Hotspot as evidenced by the warnings it prints when running on OpenJ9:

```
WARNING: Not a HotSpot compiler command compatible VM
("Eclipse OpenJ9 VM-11.0.6-internal”), compilerHints are
disabled.
```

In order to level the playing field, the System property: `java.vm.name` is set to ‘anonymous’ for all JDKs.

Benchmark Disclaimers

- Microbenchmarks are hyper-focused tests which test one very specific execution pattern
- They are not always a good predictor of program behavior.
- Most programs will not have the clean profiles of the microbenched code
- The hot path of the microbench may disappear in the context of a larger program.
- Microbenchmarks tend to only take one aspect - throughput - into account and ignore the other tradeoffs like startup and memory usage.

Hardware - The Beast

```
Model Name: Mac Pro
Model Identifier: MacPro6,1
Processor Name: 12-Core Intel Xeon E5
Processor Speed: 2.7 GHz
Number of Processors: 1
Total Number of Cores: 12
L2 Cache (per Core): 256 KB
L3 Cache: 30 MB
Memory: 64 GB
```

Hardware - The Little Monster

```
Model Name: MacBook Pro
Model Identifier: MacBookPro11,1
Processor Name: Intel Core i7
Processor Speed: 2.8 GHz
Number of Processors: 1
Total Number of Cores: 2
L2 Cache (per Core): 256 KB
L3 Cache: 4 MB
Memory: 16 GB
```

Compared JDK/JVMs

JDK/JVM combinations tested:

1. Oracle JDK 11 (v11.0.6)
2. GraalVM Enterprise Edition (v19.3.1)
3. GraalVM Community Edition (v19.3.1)
4. AdoptOpenJDK 11 w/Hotspot (v11.0.6 +10)
5. AdoptOpenJDK 11 w/OpenJ9 (v11.0.6 +10, 0.18.1)
6. OpenJDK 11 (v11.0.2)
7. OpenJDK 11 embedded Graal JVMCI (v11.0.2)
8. GraalVM Enterprise Edition C2 Compiler (v19.3.1)

JDK Flags - Part 1

In order to run the embedded Graal compiler in OpenJDK 11:

- +XX:+UnlockExperimentalVMOptions
- +XX:+EnableJVMCI
- +XX:+UseJVMCICompiler

JDK Flags - Part 2

In order to run the C2 compiler in GraalVM EE v19.3.1:

- +XX:+UnlockExperimentalVMOptions
- +XX:-UseJVMCICompiler

Notice the minus before the UseJVMCICompiler

JDK Flags - Part 3

No hints for heap sizes:

- Collection with size 1_000_000 (1 million items)
- Collection with size 1000 (1 thousand items)

Setting Xmx and Xms to 1024M (to ignore heap growth rates)

- Collection with size 1_000_000 (1 million items)
- Collection with size 1000 (1 thousand items)

Benchmark Math

Some relevant numbers in terms of what is run:

Benchmark Math

Some relevant numbers in terms of what is run:

- 7 Benchmark classes run for 8 JDK/JVMs = 56 maven executions
- Each class for each JDK/JVM takes ~1h ≈ 56 hours

Benchmark Math

Some relevant numbers in terms of what is run:

- 7 Benchmark classes run for 8 JDK/JVMs = 56 maven executions
- Each class for each JDK/JVM takes ~1h ≈ 56 hours

- Mode.Throughput on collections, seeded with different sizes

Benchmark Math

Some relevant numbers in terms of what is run:

- 7 Benchmark classes run for 8 JDK/JVMs = 56 maven executions
- Each class for each JDK/JVM takes ~1h ≈ 56 hours

- Mode.Throughput on collections, seeded with different sizes

- 42 benchmarks run for 8 JDK/JVMs = 336 benchmarks
- 336 benchmarks run in 2 forks = 672 executions

Benchmarks on IntList operations

Three operations were benchmarked on primitive collections:

Benchmarks on IntList operations

Three operations were benchmarked on primitive collections:

- Filter: Filter in all even numbers (6 benchmarks)

Benchmarks on IntList operations

Three operations were benchmarked on primitive collections:

- Filter: Filter in all even numbers (6 benchmarks)
- Sum: Sum of all integers in the collection (6 benchmarks)

Benchmarks on IntList operations

Three operations were benchmarked on primitive collections:

- Filter: Filter in all even numbers (6 benchmarks)
- Sum: Sum of all integers in the collection (6 benchmarks)
- Transform: Multiply each integer by 2 (6 benchmarks)

Benchmarks on Object operations

Four operations were benchmarked on non-primitive collections:

Benchmarks on Object operations

Four operations were benchmarked on non-primitive collections:

- Filter: Filter all Persons with height between 60 and 72 inches (7 benchmarks)

Benchmarks on Object operations

Four operations were benchmarked on non-primitive collections:

- Filter: Filter all Persons with height between 60 and 72 inches (7 benchmarks)
- FilterAndGroup: Filter all Persons less than 72 inches tall, then group by age (7 benchmarks)

Benchmarks on Object operations

Four operations were benchmarked on non-primitive collections:

- Filter: Filter all Persons with height between 60 and 72 inches (7 benchmarks)
- FilterAndGroup: Filter all Persons less than 72 inches tall, then group by age (7 benchmarks)
- IntSummaryStatistics: Summary statistics by age (integer) of all Person instances (5 benchmarks)

Benchmarks on Object operations

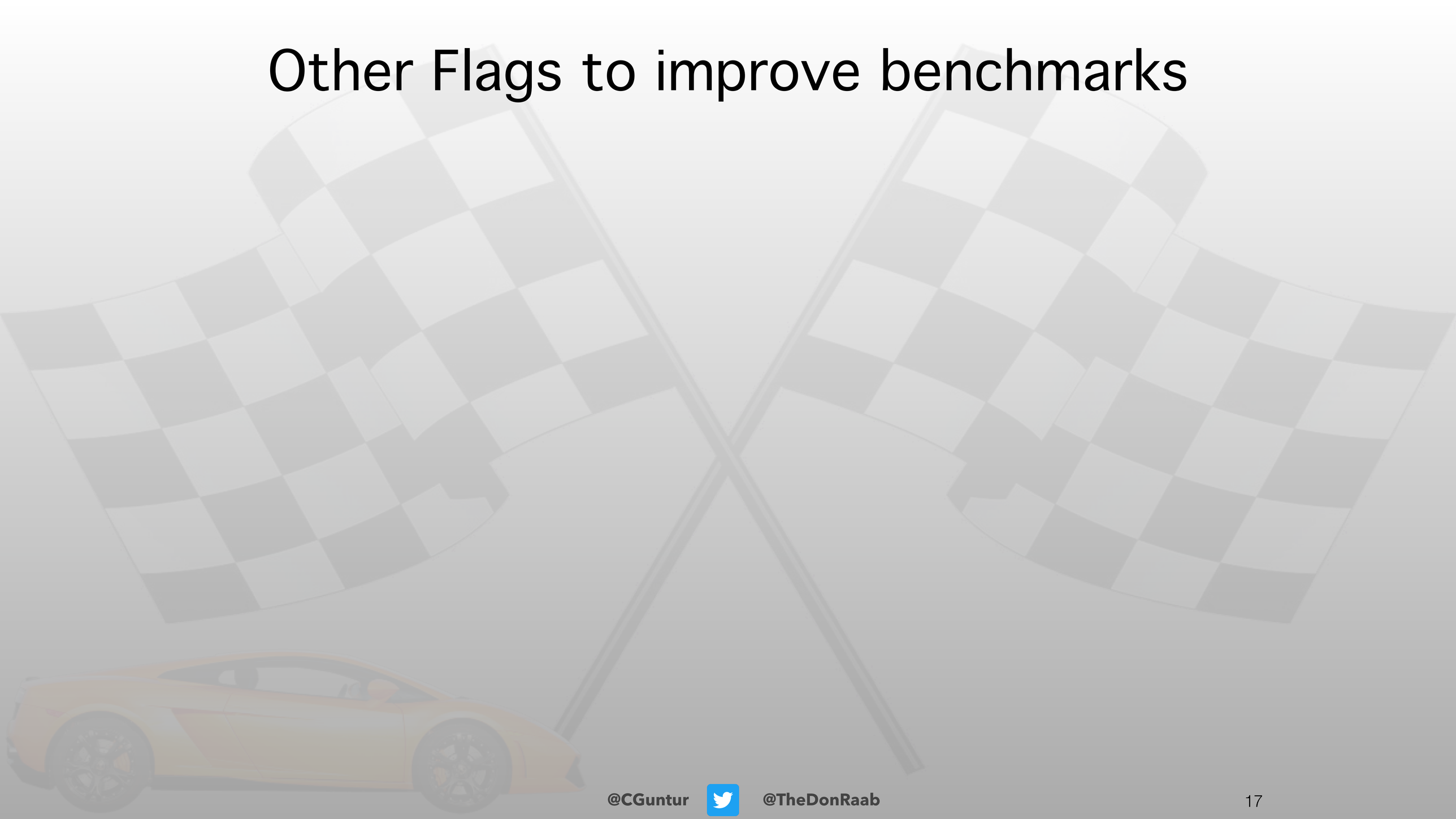
Four operations were benchmarked on non-primitive collections:

- Filter: Filter all Persons with height between 60 and 72 inches (7 benchmarks)
- FilterAndGroup: Filter all Persons less than 72 inches tall, then group by age (7 benchmarks)
- IntSummaryStatistics: Summary statistics by age (integer) of all Person instances (5 benchmarks)
- CombinedSummaryStatistics: Summary statistics on height (double), weight (double) and age (integer) (5 benchmarks)

HANDS ON !


eclipse collections
eclipse collections
eclipse collections

Other Flags to improve benchmarks



Other Flags to improve benchmarks

- `-Xjit:acceptHugeMethods,scratchSpaceLimit=1048576`

Other Flags to improve benchmarks

- `-Xjit:acceptHugeMethods,scratchSpaceLimit=1048576`
- `-Xshareclasses:none`

Other Flags to improve benchmarks

- `-Xjit:acceptHugeMethods,scratchSpaceLimit=1048576`
- `-Xshareclasses:none`

Other Flags to improve benchmarks

- `-Xjit:acceptHugeMethods,scratchSpaceLimit=1048576`
- `-Xshareclasses:none`

Other Flags to improve benchmarks

- `-Xjit:acceptHugeMethods,scratchSpaceLimit=1048576`
- `-Xshareclasses:none`

Other Flags to improve benchmarks

- `-Xjit:acceptHugeMethods,scratchSpaceLimit=1048576`
- `-Xshareclasses:none`

JMH provides custom compiler options to OpenJDK, HotSpot, Graal and select Zing JVMs using the -XX:CompileCommandFile= flag