Compile ahead of time. It's fine?

Hotspot & AOT

Java
Your
Next
(Cloud)

Dmitry Chuyko

Java SE Performance Team
April 4, 2017

Program Agenda

1. JEP 295 in JDK 9
2. Generated Library
3. External Tools
4. Performance
5. Future Directions

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

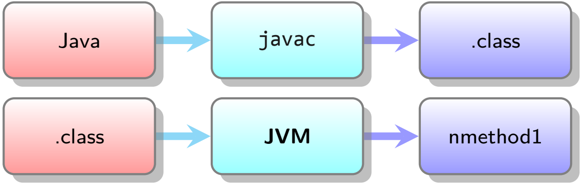
JEP 295 in JDK 9

AOT 9: Components

- JEP 295: Ahead-of-Time Compilation
  http://openjdk.java.net/jeps/295
  JDK 9 EA build 150
- JEP 243: Java-Level JVM Compiler Interface
  http://openjdk.java.net/jeps/243
- Graal Compiler
  https://github.com/graalvm/graal-core

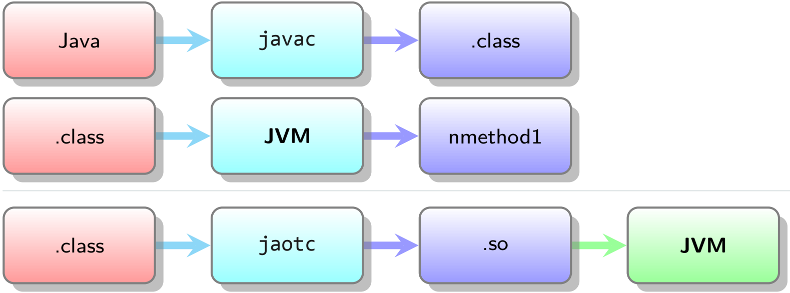
AOT 9: Workflow

Regular



AOT 9: Workflow

Pre-compilation



Java

javac

.class

.class

JVM

nmethod1

.class

jaotc

.so

JVM

AOT 9: Targeted Problems

- Application Warm-up
  - Startup Time
  - Time to Performance
- Steady state
  - Peak Performance
  - Application Latency
- Complex case
  - Bootstrapping (meta-circular implementations)
- Possible impact
  - Density
  - Power Consumption

AOT 9: Solutions

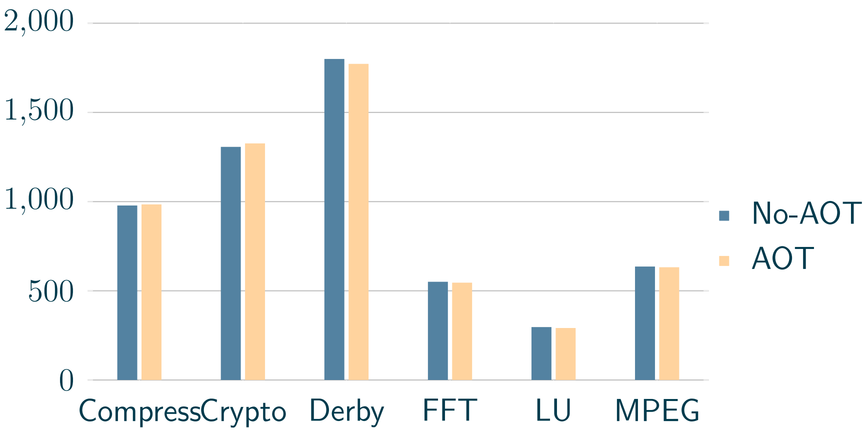
- Pre-compile initialization code
  - No interpreter for class loading, initializers etc.
  - Spare resources for compilation
  - May stay at AOT
- Pre-compile critical code
  - Start with much better than interpreter performance
  - Spare resources for compilation
  - May stay at AOT
- Collect same profiling info as tier 2
  - Reach peak performance

AOT 9: Measurements

- JDK 9 EA build 162
- Linux x64
- G1
- Compressed oops
- Dedicated server hardware or small machine

AOT 9: AOT vs. JIT

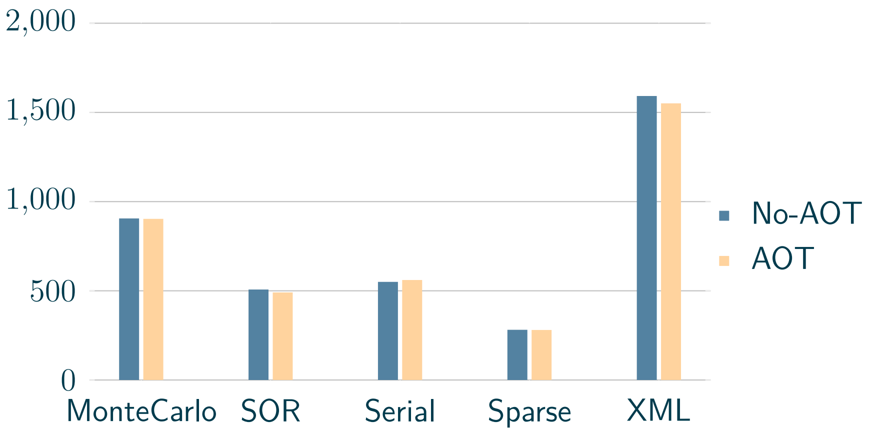
naïve



SPECjvm2008
G1
Tiered AOT
of java.base
Linux x64

AOT 9: AOT vs. JIT

naïve



SPECjvm2008
G1
Tiered AOT
of java.base
Linux x64

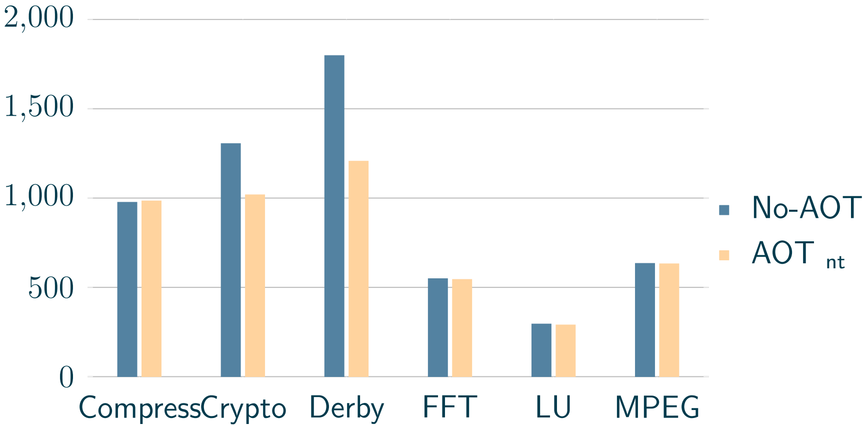
AOT 9: Tiered AOT throughput

Not so useless

- ✓ It works
- ✓ Ensure peak performance in steady state
- ✓ There may be differencies
  - Treated as bugs
  - Ignored

AOT 9: AOT vs. JIT

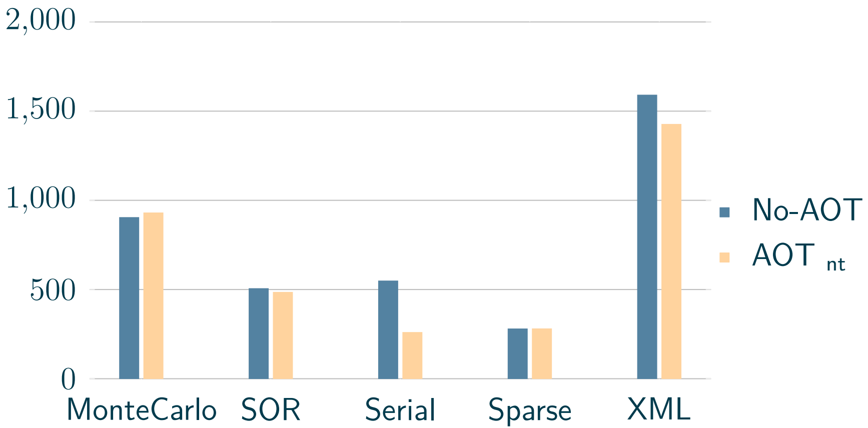
Frustrating



SPECjvm2008
G1
Non-tiered AOT of java.base
Linux x64

AOT 9: AOT vs. JIT

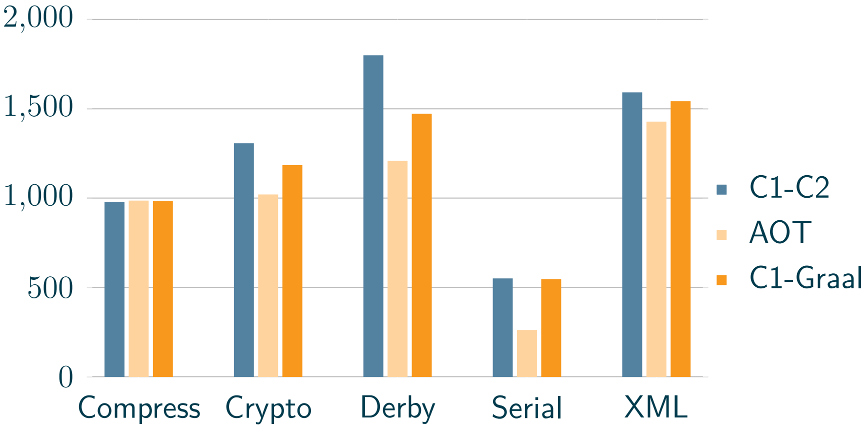
Frustrating



SPECjvm2008
G1
Non-tiered AOT of java.base
Linux x64

AOT 9: Is it Graal?

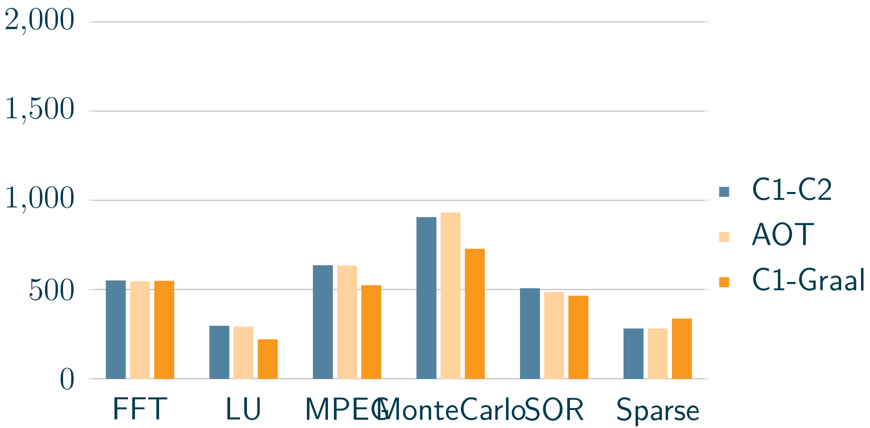
Ones regressed with AOT may not differ



SPECjvm2008
G1
Non-tiered AOT
of java.base
Linux x64

AOT 9: Is it Graal?

Ones may only differ with Graal as JIT



SPECjvm2008
G1
Non-tiered AOT
of java.base
Linux x64

AOT 9: AOT throughput

- Benchmarks regressed with AOT may not differ with Graal as JIT
- Benchmarks may only differ with Graal as JIT
- Same for other large benchmarks (e.g. SPECjbb)
- Same for many JVM micro-benchmarks
- It's common to see NN% difference

Generated Library

Generated Libraries: Auto-loaded

Original & striped, compressed oops

	jmod	Methods	Tiered G1	NT G1	Tiered Par
base	19M	50673	416M / 286M	318M / 201M	395M / 264M
logging	118K	532	3.8M / 2.6M	2.9M / 1.8M	3.6M / 2.3M
nashorn	2.2M	11865	84M / 54M	64M / 37M	79M / 49M
jvmci	386K	1750	12M / 8.5M	8.9M / 5.8M	12M / 7.6M
graal	5.5M	18166	163M / 104M	127M / 73M	154M / 95M
javac	6.3M	12446	115M / 75M	91M / 55M	109M / 69M

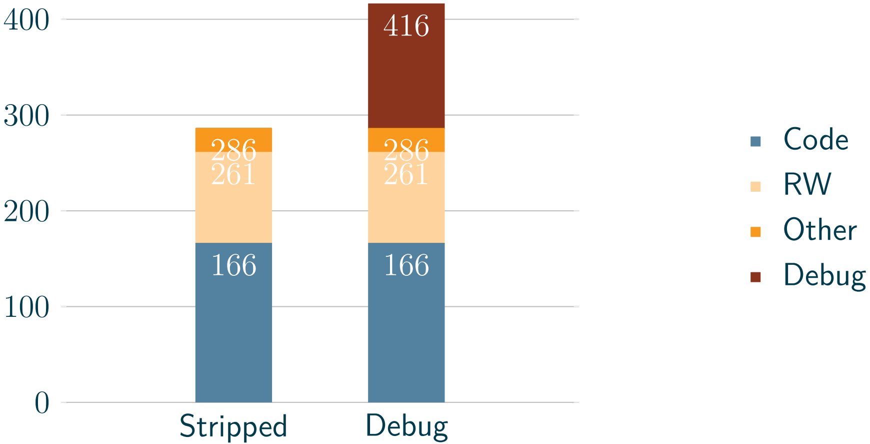
Generated Libraries: Basic subsets

Original & striped, compressed oops

	Methods	Tiered G1
java.base-CDS	22375	163M / 112M
java.base-Hello	615	5.3M / 3.5M
hello	2	99K / 76K

Generated Libraries: libjava.base-coop.so

```
readelf -S, size -A -d
```



Generated Libraries: Shared library

- Shareable
- Native debug information
- Code
- Metadata
  - .so → VM linkage
  - VM → .so linkage
  - Runtime support

Generated Libraries: Hello World

```
./objconv -dh any-aot.so.dbg | ...
```

```
.hash                Symbol hash table
.dynsym              Dynamic linker symbol table
.dynstr              String table
.rela.dyn            Relocation w addends
.text                Program data
.metaspace.names     Program data
.klasses.offsets     Program data
.methods.offsets     Program data
.klasses.dependencies Program data
.stubs.offsets       Program data
.header              Program data
.code.segments       Program data
.method.constdata    Program data
```

```
.config              Program data
.eh_frame            Program data
.dynamic             Dynamic linking info
.metadata.got        Program data
.method.metadata     Program data
.hotspot.linkage.got Program data
.metaspace.got       bss
.method.state        bss
.oop.got             bss
.shstrtab            String table

.symtab              Symbol table
.strtab              String table
```

Generated Libraries: Hello World

```
./objdump -d hello.so.dbg | ...

00000000000023a0 <test.HelloWorld.<init>()V>:
0000000000002520 <test.HelloWorld.main([Ljava/lang/String;)V>:
0000000000002b48 <M1_375_java.io.PrintStream.write(Ljava/lang/String;)V_plt.entry>:
0000000000002b5b <M1_375_java.io.PrintStream.write(Ljava/lang/String;)V_plt.jmp>:
0000000000002b68 <M1_391_java.io.PrintStream.newLine()V_plt.entry>:
0000000000002b7b <M1_391_java.io.PrintStream.newLine()V_plt.jmp>:
...
```

Generated Libraries: Hello World

```
./objdump -d hello.so.dbg | ...
0000000000002c20 <Stub<AMD64MathStub.log>>:
...
0000000000005e20 <Stub<NewInstanceStub.newInstance>>:
0000000000005f20 <Stub<NewArrayStub.newArray>>:
0000000000006020 <Stub<ExceptionHandlerStub.exceptionHandler>>:
...
0000000000007ca0 <Stub<test_deoptimize_call_int(int)int>>:
...
0000000000007d80 <plt._aot_jvmci_runtime_new_instance>:
0000000000007d88 <plt._aot_jvmci_runtime_new_array>:
0000000000007d90 <plt._aot_jvmci_runtime_exception_handler_for_pc>:
...
0000000000007e58 <plt._aot_backedge_event>:
0000000000007e60 <plt._aot_jvmci_runtime_thread_is_interrupted>:
0000000000007e68 <plt._aot_jvmci_runtime_test_deoptimize_call_int>:
```

Generated Libraries: Cold HelloWorld startup

Slow HDD. Size matters

	real	user	sys
No-AOT	1.8s	0.2s	0.0s
java.base (used)	12.5s	0.4s	0.4s
Large unused	2.1s	0.2s	0.1s
App	1.8s	0.2s	0.0s

Generated Libraries: Warm HelloWorld startup

	real	**user**	**sys**
No-AOT	0.12s	0.15s	0.02s
java.base	0.15s	0.13s	0.02s

Generated Libraries: Profiling strategies

```
jaotc -J-Dgraal.ProfileSimpleMethods=false
```

	Tiered G1	Tiered no-PSM	Non-tiered G1
java.base	416M / 286M	370M / 252M	318M / 201M

Generated Libraries: Profiling strategies

```
org.graalvm.compiler.hotspot.phases.profiling.FinalizeProfileNodesPhase

    @Override
    protected void run(StructuredGraph graph, PhaseContext context) {
        if (simpleMethodHeuristic(graph)) {
            removeAllProfilingNodes(graph);
            return;
        }

        assignInlineeInvokeFrequencies(graph);
        if (ProfileNode.Options.ProbabilisticProfiling.getValue()) {
            assignRandomSources(graph);
        }
    }
```

Generated Libraries: Profiling strategies

```
org.graalvm.compiler.hotspot.phases.profiling.FinalizeProfileNodesPhase

   private static boolean simpleMethodHeuristic(StructuredGraph graph) {
        if (Options.ProfileSimpleMethods.getValue()) {
            return false;
        }

        // Check if the graph is smallish..
        if (graph.getNodeCount() > Options.SimpleMethodGraphSize.getValue()) {
            return false;
        }

        // Check if method has loops
        if (graph.hasLoops()) {
            return false;
        }
...
```

Generated Libraries: Patching Graal

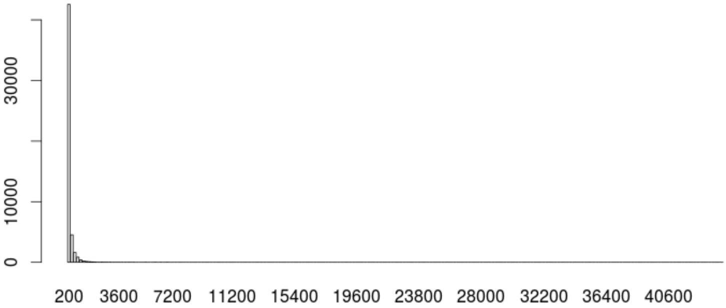
```
org.graalvm.compiler.hotspot.phases.profiling.FinalizeProfileNodesPhase

    static ExecutorService io = Executors.newSingleThreadExecutor();
    @Override
    protected void run(StructuredGraph graph, PhaseContext context) {
      int nodeCount = graph.getNodeCount();
      // int nodeCount = graph.getNodes().filter(InvokeNode.class).count(); etc.
      io.execute(() -> {
        try {
          File hist = new File("hist.csv");
          if(!hist.exists()) hist.createNewFile();
          BufferedWriter bw = new BufferedWriter(new FileWriter(hist.getName(), true));
          bw.write(Integer.toString(nodeCount)); bw.write("\n");
          bw.close();
        } catch (IOException e) { };
      });
...
```

Generated Libraries: Patching Graal

```
javac --patch-module jdk.internal.vm.compiler=. \\
 org/graalvm/compiler/hotspot/phases/profiling/FinalizeProfileNodesPhase.java

jaotc -J--patch-module -Jjdk.internal.vm.compiler=/home/tp/aot/patching \\
 -J-XX:+UseCompressedOops -J-XX:+UseG1GC -J-Xmx4g \\
 --info --module java.base --compile-for-tiered --output ignored.so
```

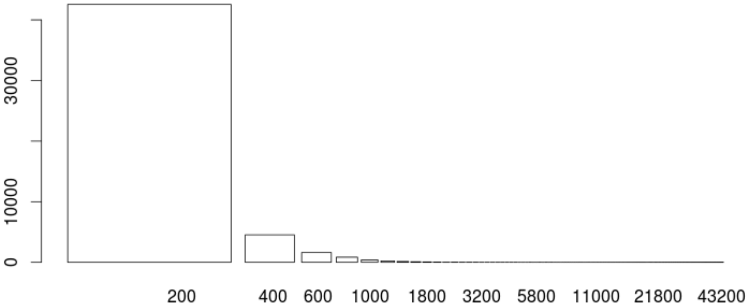
Generated Libraries: Number of nodes in method graphs

java.base



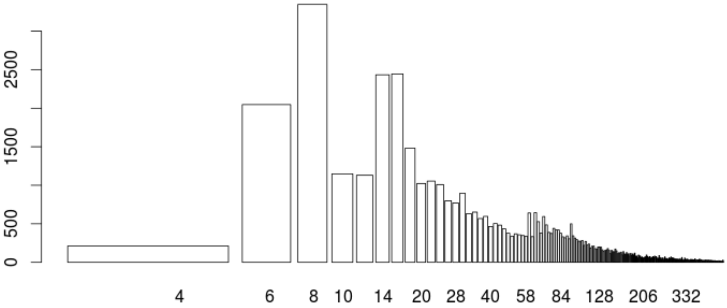
Generated Libraries: Number of nodes in method graphs

java.base



Generated Libraries: Number of nodes in method graphs

java.base



External Tools

Profiling: Flames

- CPU Flame Graphs

  http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html

- Perf a fork after warm-up

  ```
  perf record -F 399 -a -g -- javac-javac
  ```

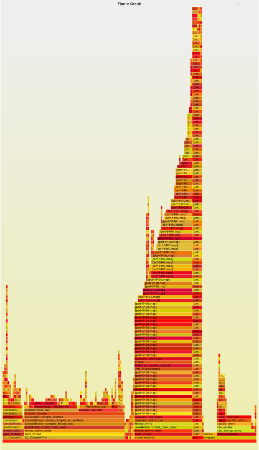
  ```
  -XX:+PreserveFramePointer
  ```

- AOT'ed modules

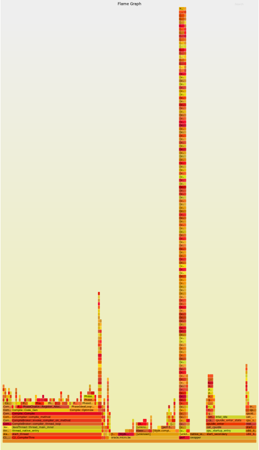
  `java.base, jdk.compiler`

- No `perf-map-agent`

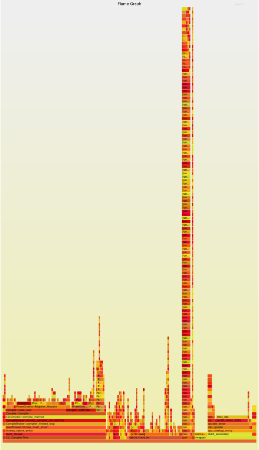
Profiling: Flames



No-AOT



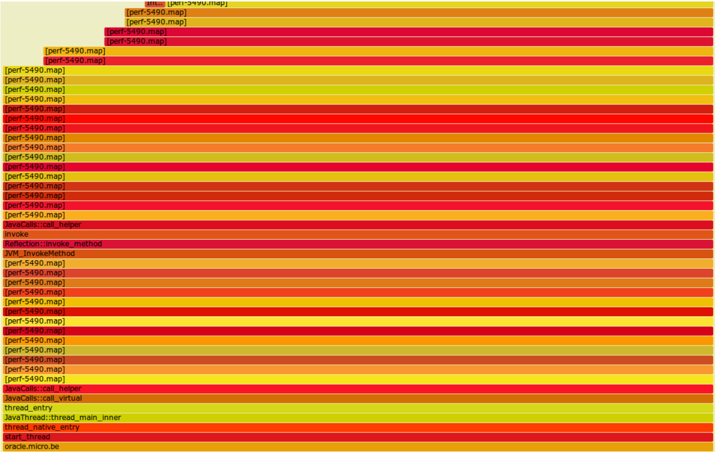
AOT without debug info



AOT with debug info

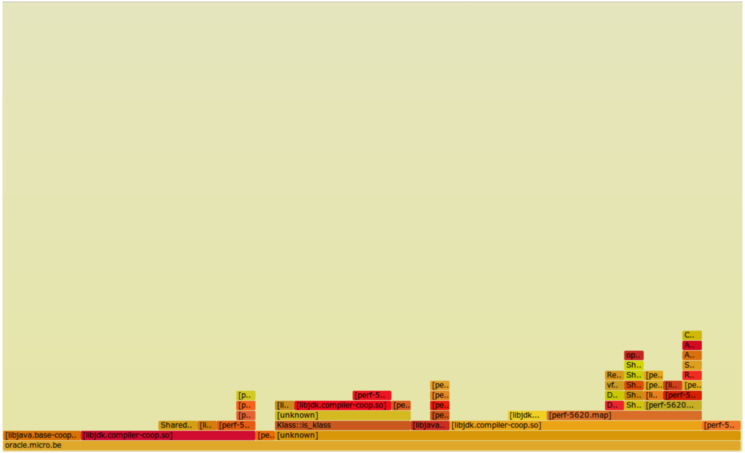
Profiling: Flames

No-AOT



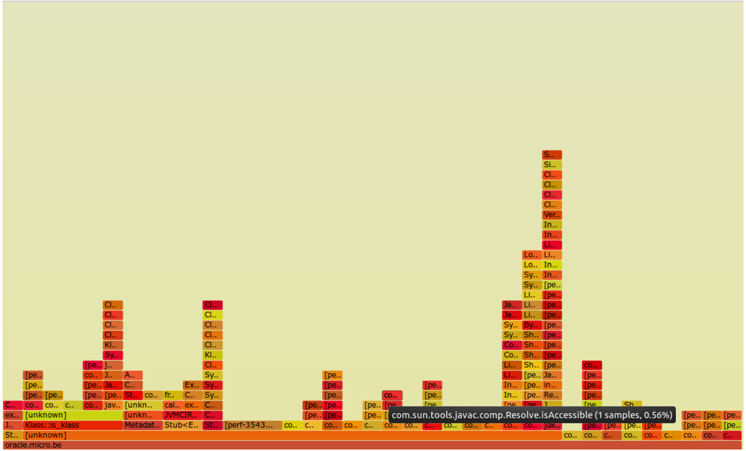
Profiling: Flames

AOT without debug info



Profiling: Flames

AOT with debug info



com.sun.tools.javac.comp.Resolve.isAccessible (1 samples, 0.56%)

Performance

Time: Startup*

	No-AOT	**java.base+app**
Javac-Hello	1.8s	−20%
Javac-Javac	17.1s	−24%

Time: Startup*

	No-AOT	java.base+app	java.base-nt+app-nt
Javac-Hello	1.8s	−20%	−38%
Javac-Javac	17.1s	−24%	−32%

* Multi-threaded (T=32)

Time: Startup*

	No-AOT	java.base+app	java.base-nt+app-nt
Javac-Hello	1.8s	−20%	−38%
Javac-Javac	17.1s	−24%	−32%

* Multi-threaded (T=32)

Single-threaded (T=1):

	No-AOT	java.base+app	java.base-nt+app-nt
Javac-Hello	0.5s	−11%	+2%
Javac-Javac	4.5s	+8%	+10%

Warmup: Contended

- No profiling → no contention
- -J-Dgraal.ProbabilisticProfiling=true
  - Tuning
  - Switch off when $T = 1$
- -J-Dgraal.ProfileSimpleMethods=true
  - Pick strategy to not profile

Warmup: Contended

ProbabilisticProfiling

HotSpotAOTProfilingPlugin.java

```
-J-Dgraal.TierAInvokeNotifyFreqLog=13
-J-Dgraal.TierABackedgeNotifyFreqLog=16
-J-Dgraal.TierAInvokeProfileProbabilityLog=8
-J-Dgraal.TierABackedgeProfileProbabilityLog=12
```

globals.hpp

```
-XX:Tier2InvokeNotifyFreqLog=11
-XX:Tier2BackedgeNotifyFreqLog=14
```

- Profile method
- Notify counters
- Logarithm of denominator

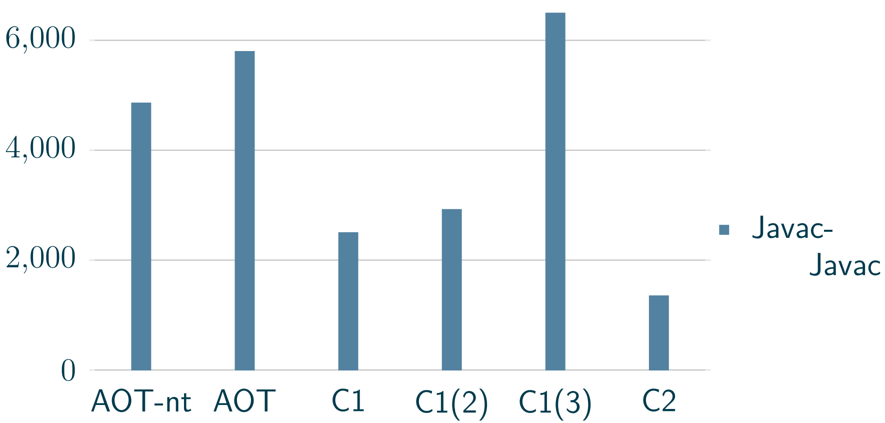
Time: Startup & Post-warmup

- C1, C1(2), C1(3)

  `-XX:TieredStopAtLevel=`k
- C2
- AOT-nt. java.base-nt & app-nt
- AOT. java.base & app

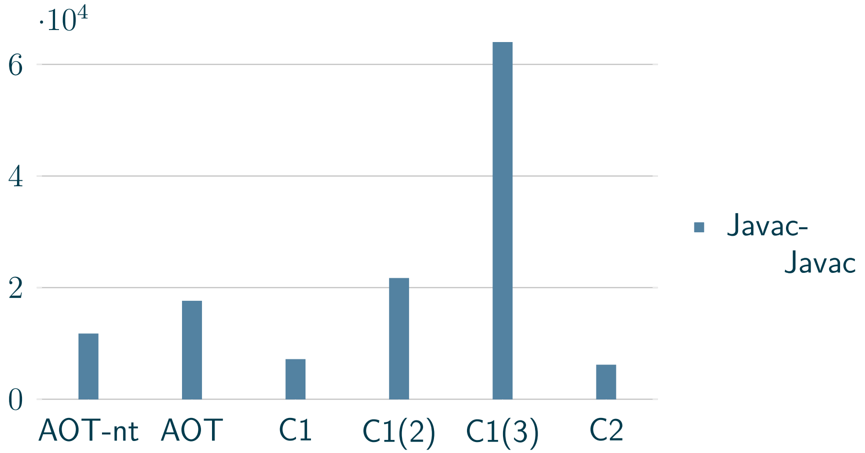
```
-XX:Tier3AOTInvocationThreshold=2000000000

-XX:Tier3AOTMinInvocationThreshold=2000000000

-XX:Tier3AOTCompileThreshold=2000000000

-XX:Tier3AOTBackEdgeThreshold=2000000000

-XX:CICompilerCount=2 -XX:TieredStopAtLevel=2
```

Time: Startup & Post-warmup



$T = 1$

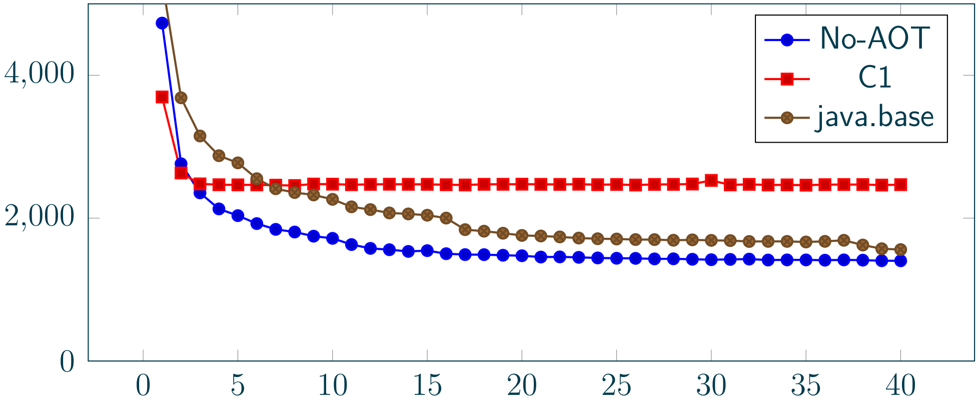
Time: Post-warmup



$T = 32$

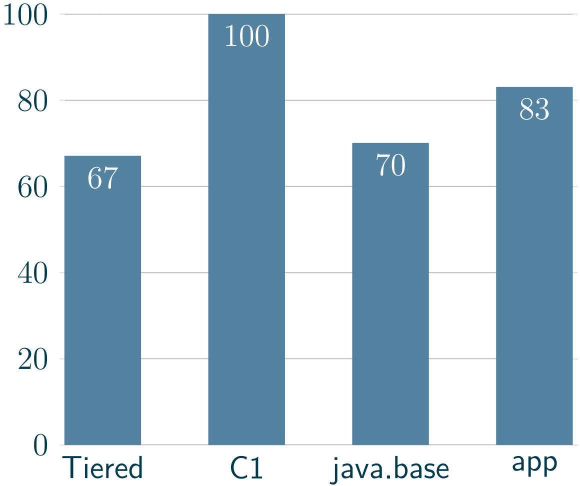
Warmup: Single threaded

Javac-javac, tiered



Warmup: Time to iterate

Javac-javac, tiered



Warmup: Tiered

```
vm/runtime/globals.hpp
```

```
-XX:Tier3AOTInvocationThreshold=10000
-XX:Tier3AOTMinInvocationThreshold=1000
-XX:Tier3AOTCompileThreshold=15000
-XX:Tier3AOTBackEdgeThreshold=120000
```

```
-XX:Tier3InvocationThreshold=200
-XX:Tier3MinInvocationThreshold=100
-XX:Tier3CompileThreshold=2000
-XX:Tier3BackEdgeThreshold=60000
```

- Thresholds are different
- Delay tier 3 on startup
- No qualitative effect on long warmup

Throughput: Measurement

What may be interesting

- AOT'ed code calling other code
- AOT'ed code touching other data
- java.base

```
@State(Thread)
public class OpsBench {
  @Benchmark
  public Result maybeFromAot() {
    return OpsClass1.doOp(<args>);
  }
}
```

```
@CompilerControl(DONT_INLINE)
public class OpsClass1 {
  public static Result doPr(String s) {
    // May use OpsClass2, may be .so
  }
}
```

Throughput: Simple method calls

	VM	**.so→VM**	**VM→.so**	**1.so→2.so**	**.so**
instance final	3.1	3.5	3.1	3.5	3.5
static direct	2.7	3.1	2.7	3.1	3.1
static indirect self	4.7	3.1	4.7	3.1	3.1
static indirect other	4.7	3.5	4.6	3.5	3.5
infra	0.4				

ns/op, ±1ns

- It's hard to measure directly

Throughput: Simple method calls

VM→.so, perfasm

```
....[Distribution by Source]..............
46.57%  45.21%  c2, level 4
25.62%  25.60%  c1, level 1
25.62%  27.22%      lib2.so
 0.75%   0.71%       kernel
 0.69%   0.61%    libjvm.so

....[Hottest Methods (after inlining)]....
37.83%  41.27%  c2, level 4  benchmarks.generated.CallBench_invokeStaticOther_jmhTest::in
25.60%  25.17%  c1, level 1  benchmarks.TargetClass1::staticThatTarget, version 543
24.90%  26.19%      lib2.so  benchmarks.TargetClass2.staticEmptyTarget()V
 8.94%   5.80%  c2, level 4  benchmarks.generated.CallBench_invokeStaticOther_jmhTest::in
 1.72%   0.79%       kernel  [unknown]
 0.08%   0.18%    libjvm.so  ElfSymbolTable::lookup
```

Throughput: Read data

	VM	**.so→VM**
Read static int	6.0	6.9
Read length of static string	6.7	8.7

ns/op, ±1ns

Throughput: Read data

String length, perfnorm

	VM	**.so→VM**
Time, ns/op	6.5	8.8
L1 dcache loads	16.4	25.6
Branches	6.1	12.3
Cycles	17.3	23.5
Instructions	39.8	64.4

Throughput: Read data

String length, perfasm

```
....[Hottest Region 2].....................
c2, level 4,
benchmarks.AccessClass1::staticThatStrlen,
version 544 (52 bytes)
```

```
....[Hottest Region 1].....................
lib1.so,
benchmarks.AccessClass1.staticThatStrlen()I
(159 bytes)
```

Throughput: Read data

String length, asm

Constants in C2

```
0x00007f4d596b7aac: mov    $0x8eff8e70,%r10
     ;   {oop(a &apos;java/lang/Class&apos;{0x000000008eff8e70}
      = &apos;benchmarks/AccessClass2&apos;)}
....
0x00007f4d596b7adb: callq  0x00007f4d51c0dc00  ; ImmutableOopMap{}
                           ;*invokevirtual length {reexecute=0 rethrow=0 return_oop=0}
```

Checks in AOT

```
2541:    mov    0x20fad8(%rip),%rcx
# 212020 <got.init.Lbenchmarks/AccessClass2;>
2548:    test   %rcx,%rcx
254b:    je     25e2
<benchmarks.AccessClass1.staticThatStrlen()I+0xc2>
2551:    mov    0x20fad0(%rip),%rcx
# 212028 <got.L/benchmarks/AccessClass2;>
```

Latency: Garbage collection

With AOT

- Some additional GC work
- No sensitive impact on mean
- No sensitive impact on max
- Same distributions

Startup: Applications

WLS

base_domain System Classloader	no-AOT no-CDS	java.base no-CDS	no-AOT AppCDS	java.base AppCDS
Startup	11.4s	−17%	−33%	−48%
Footprint [x1]				
resident	478 M	+25%	−3%	+25%
unique	466 M	−6%	−15%	−18%
Footprint [x10]				
total	4652 M	−3%	−11%	−13%

Startup: Applications

Jetty

	No-AOT	java.base	java.base-nt
Jetty	0.5s	−15%	−22%

Startup: Graal bootstrap

$T = 1$

	No-AOT	**java.base**	**java.base+graal+jvmci**
Javac-Hello	0.8s	−29%	−29%
Jetty	0.5s	0%	0%
Javac-Javac	4.6s	−6%	−5%
Javadoc-Small	2.7s	−2%	+2%

Future Directions

Future Directions: More platforms

- Other *NIX with ELF
- PEF (macOS)
- PE (Windows)
- ARM64 port

Future Directions: Less harmful

- Smaller footprint
- Multi-mode
- Cross-AOT

Future Directions: Features convergence

- Solve class data access problem
  - CDS
  - AppCDS
  - Shared strings
- Boilerplate
  - AOT of pre-generated stuff
- Product features
  - WLS
- Cloud
  - Containers

Future Directions: Java on Java

Goals

- Simple maintenance
- Faster development
- Better security
- Embeddable VM

Future Directions: Java on Java

Currently

- Class library
- Method handles
- Graal/JVMCI
- AOT

Future Directions: Java on Java

Possibly more

- Graal as JIT
  - Replacement for C2, then C1, then interpreter
- Runtime
  - Class file parser
  - Verifier
  - Reflection
  - Stub generation
- Compiler
  - Method liveness

Future Directions: Project Metropolis

- JDK 10 based
- *System* Java
- Translated parts of Hotspot
- Graal
- AOT

Future Directions: Java on Java

Dependencies

- Java ↔ native interop
  - Project Panama
- Operate pointer-poor (flat) data
  - Project Valhalla
- Bootstrap code
  - AOT