

JIT Compilation with Graal

Journey of a Java Program

Swapnil Gaikwad, Alan Hayward
Arm, Manchester

 @SwapnilGaikwad



Agenda

- Lifecycle of a Java program
- Interpreter
- JIT Compilation
 - Graal Compiler
- Compiler Optimisations
- Summary

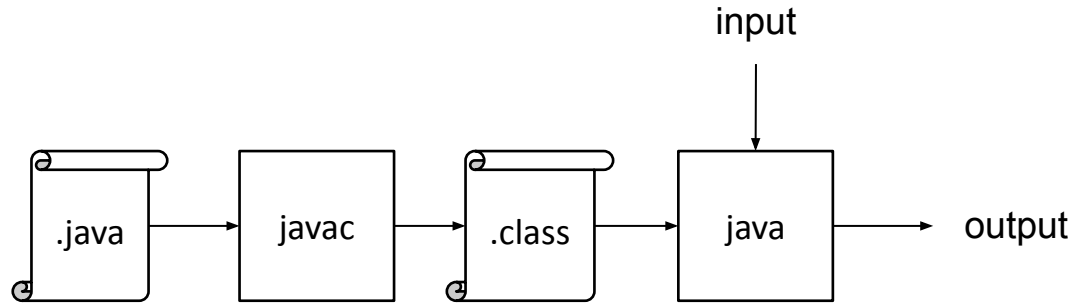


<https://github.com/SwapnilGaikwad/devoxx22>

Lifecycle of a Java Program

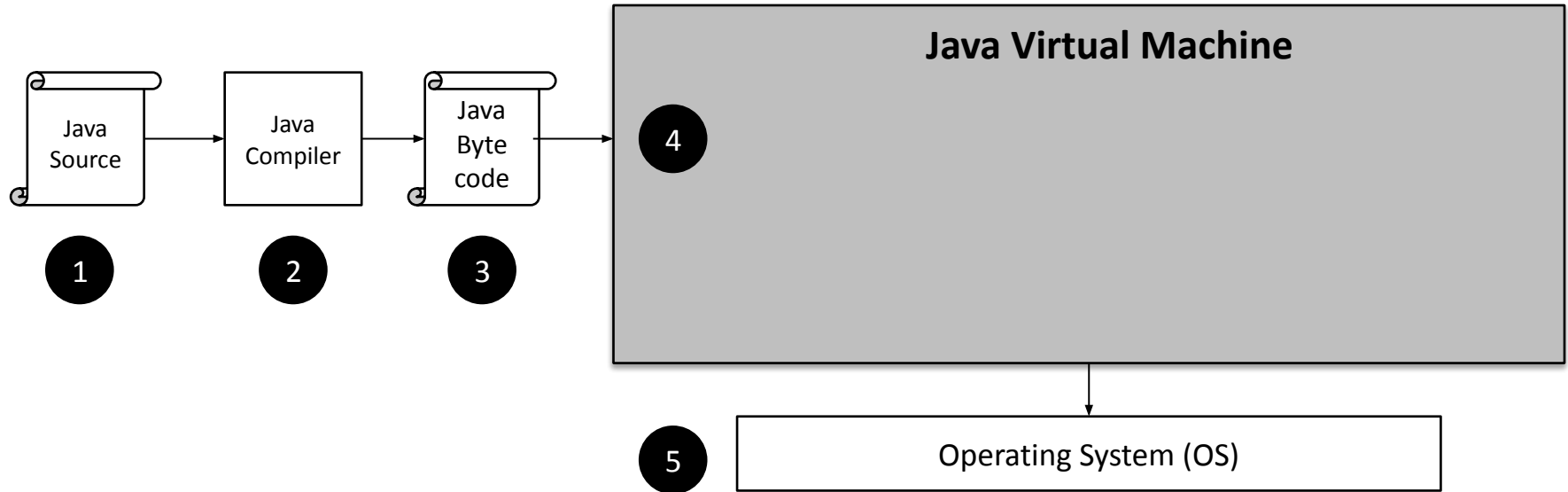
```
$ javac MyTest.java  
$ java MyTest
```

Lifecycle of a Java Program



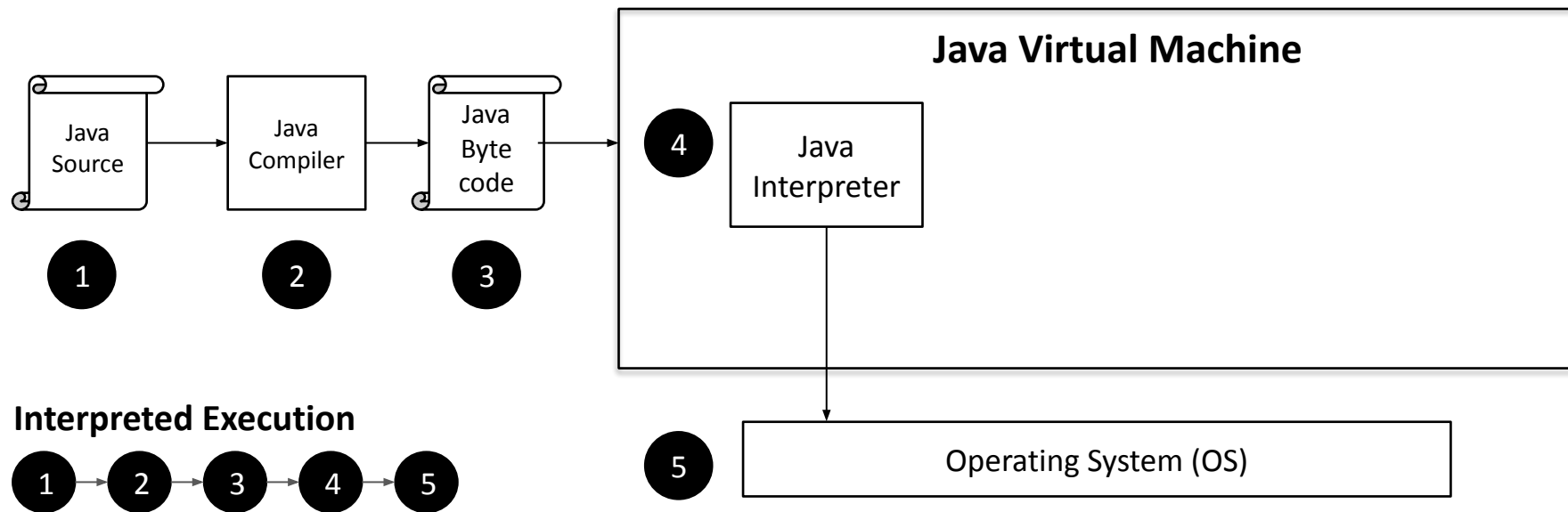
```
$ javac MyTest.java  
$ java MyTest
```

Java Compilation Approach

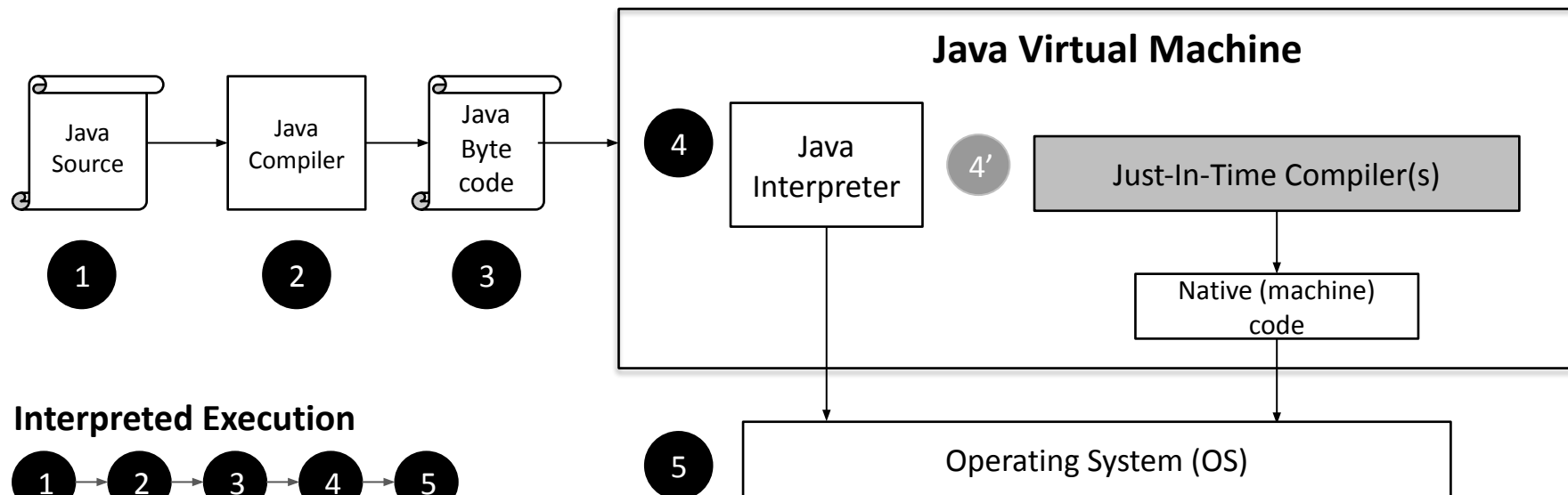


- Java uses hybrid approach of both compiler and interpreter

JVM Overview



JVM Overview



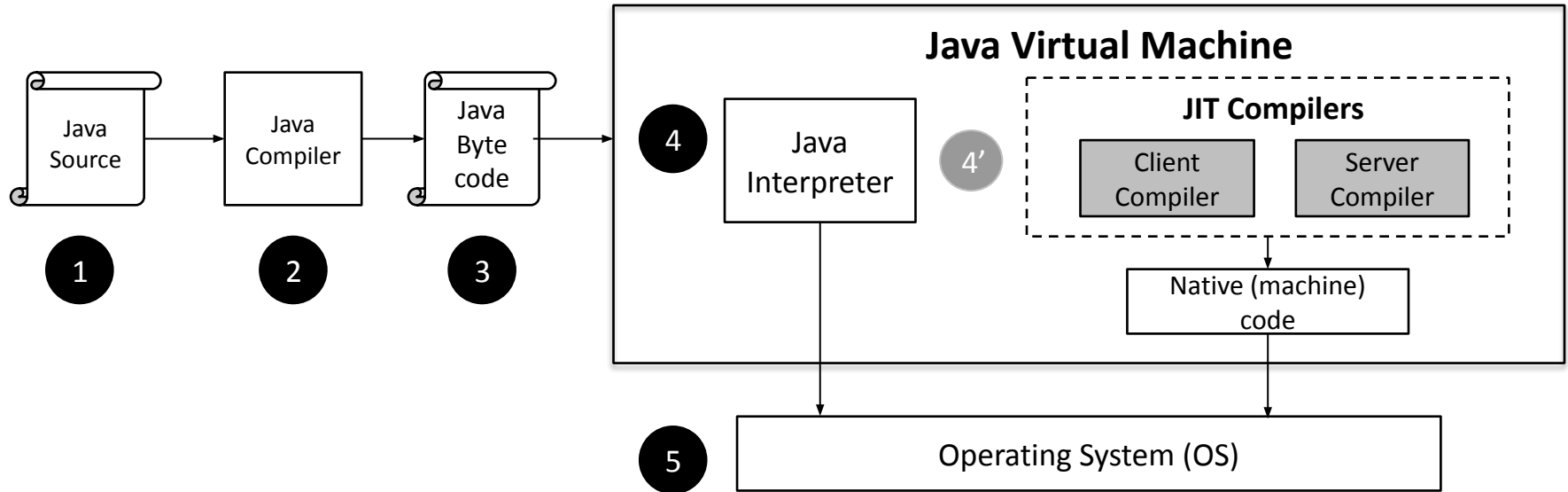
Interpreted Execution



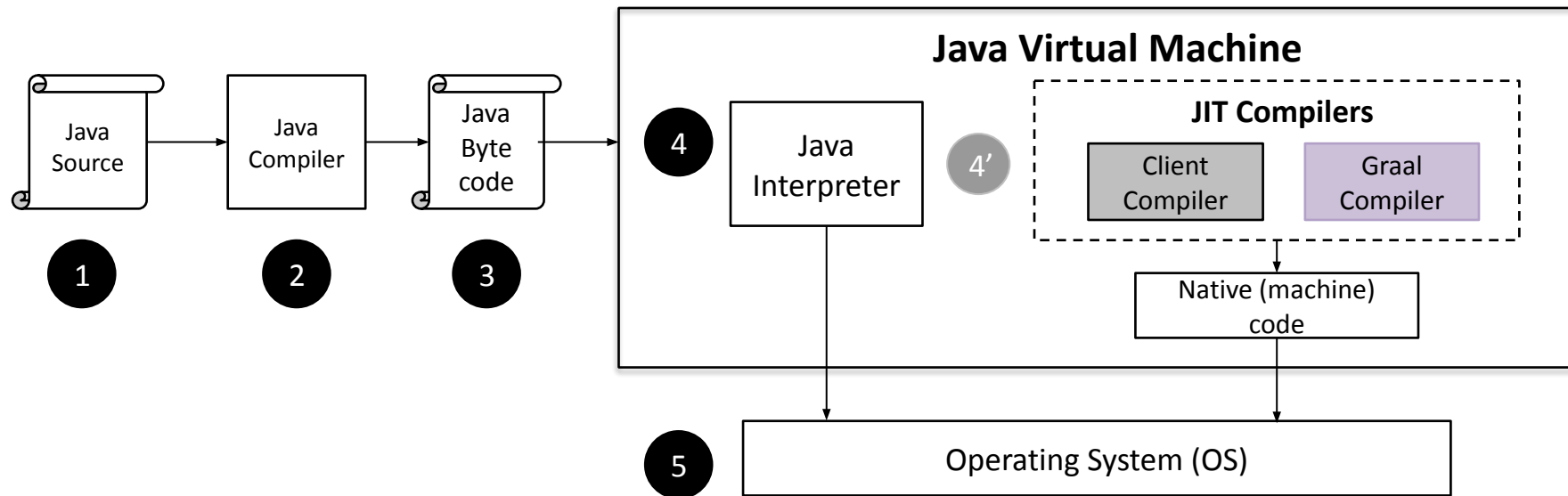
JIT Compiled Execution



HotSpot JVM Overview



Graal VM Overview



A Simple Java Program

```
public class MyTest {  
    public static int myInc(int a) {  
        return a + 1;  
    }  
  
    public static void main(String[] args) {  
        MyTest.myInc(5);  
    }  
}
```

Interpreter: Bytecodes

...

```
public static int myInc(int);
```

Code:

0: iload_0

1: iconst_1

2: iadd

3: ireturn

...

```
$ javap -c MyTest.class
```

Interpreter: Bytecodes

```
public static int myInc(int);
```

Code:

```
0: iload_0  
1: iconst_1  
2: iadd  
3: ireturn
```

```
Call myInc(5)
```

Interpreter: Bytecodes

```
public static int myInc(int);
```

Code:

```
0: iload_0  
1: iconst_1  
2: iadd  
3: ireturn
```

Load the first local variable



5

Interpreter: Bytecodes

```
public static int myInc(int);
```

Code:

```
0: iload_0  
1: iconst_1  
2: iadd  
3: ireturn
```

Push constant 1

1
5

Interpreter: Bytecodes

```
public static int myInc(int);
```

Code:

```
0: iload_0  
1: iconst_1  
2: iadd  
3: ireturn
```

Pop top two values, add them and
push result on the stack



6

Interpreter: Bytecodes

```
public static int myInc(int);
```

Code:

```
0: iload_0  
1: iconst_1  
2: iadd  
3: ireturn
```

Return int from method

6

Performance

- Interpreted version
- JIT Compiled with C1 compiler
- JIT Compiled with Graal compiler

Benchmark	Mode	Cnt	Score	Error	Units
MyBenchmark.interpreter	avgt	3	41469.329 ±	1390.876	ns/op
MyBenchmark.tier1only	avgt	3	614.950 ±	16.311	ns/op
MyBenchmark.tierAll	avgt	3	389.108 ±	8.780	ns/op

Interpreter: Assembly for a bytecode

```
public static int myInc(int);
```

Code:

```
0: iload_0  
1: iconst_1  
2: iadd  
3: ireturn
```

```
-----  
iload_0  26 iload_0  [0x0000ffffd2ea6200, 0x0000ffffd2ea6240] 64 bytes  
  
0x0000ffffd2ea6200: str      x0, [x20,#-8]!  
0x0000ffffd2ea6204: b 0x0000ffffd2ea6228  
0x0000ffffd2ea6208: str      s0, [x20,#-8]!  
0x0000ffffd2ea620c: b 0x0000ffffd2ea6228  
0x0000ffffd2ea6210: str      d0, [x20,#-16]!  
0x0000ffffd2ea6214: b 0x0000ffffd2ea6228  
0x0000ffffd2ea6218: str      xzr, [x20,#-8]!  
0x0000ffffd2ea621c: str      x0, [x20,#-8]!  
0x0000ffffd2ea6220: b 0x0000ffffd2ea6228  
0x0000ffffd2ea6224: str      x0, [x20,#-8]!  
0x0000ffffd2ea6228: ldr      x0, [x24]  
0x0000ffffd2ea622c: ldrb     w8, [x22,#1]!  
0x0000ffffd2ea6230: add      w9, w8, #0x400  
0x0000ffffd2ea6234: ldr      x9, [x21,w9,uxtw #3]  
0x0000ffffd2ea6238: br       x9  
0x0000ffffd2ea623c: nop
```

Performance

Benchmark	Mode	Cnt	Score	Error	Units
MyBenchmark.interpreter	avgt	3	41469.329 ±	1390.876	ns/op
MyBenchmark.tier1Only	avgt	3	614.950 ±	16.311	ns/op
MyBenchmark.tierAll	avgt	3	389.108 ±	8.780	ns/op

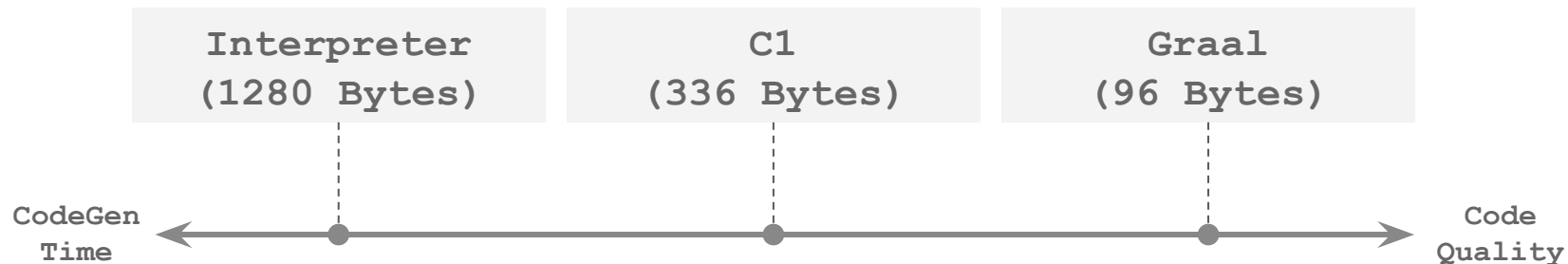
Interpreter
(1280 Bytes)

C1
(336 Bytes)

Graal
(96 Bytes)

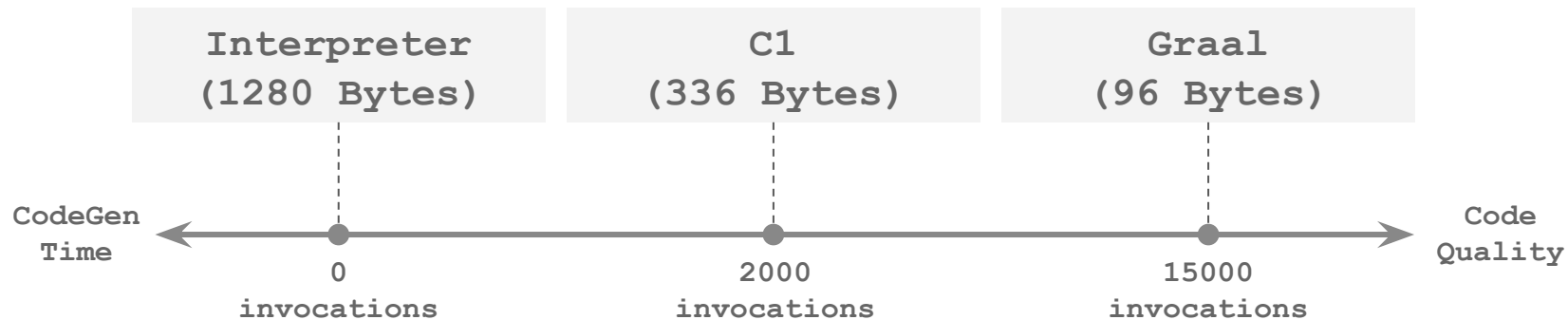
Performance

Benchmark	Mode	Cnt	Score	Error	Units
MyBenchmark.interpreter	avgt	3	41469.329 ±	1390.876	ns/op
MyBenchmark.tier1Only	avgt	3	614.950 ±	16.311	ns/op
MyBenchmark.tierAll	avgt	3	389.108 ±	8.780	ns/op

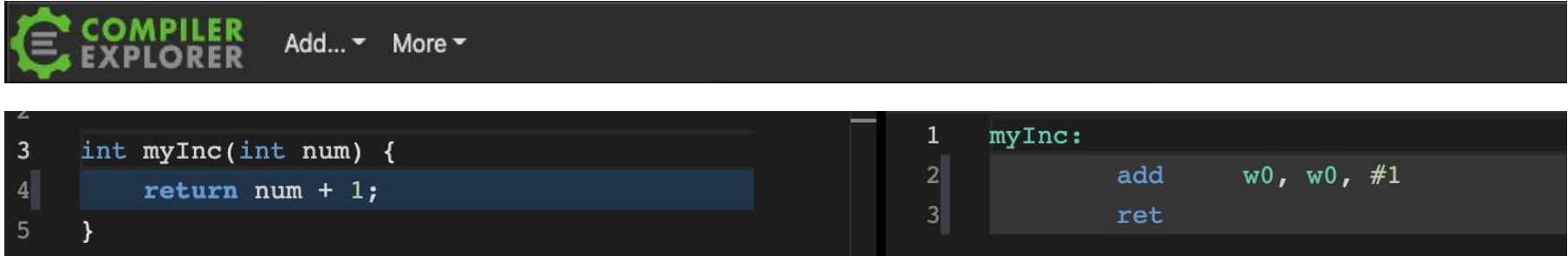


Performance

Benchmark	Mode	Cnt	Score	Error	Units
MyBenchmark.interpreter	avgt	3	41469.329 ±	1390.876	ns/op
MyBenchmark.tier1Only	avgt	3	614.950 ±	16.311	ns/op
MyBenchmark.tierAll	avgt	3	389.108 ±	8.780	ns/op



Assembly in C



The image shows the Compiler Explorer interface. The top bar has the 'COMPILER EXPLORER' logo and two dropdown menus labeled 'Add...' and 'More...'. The main area is split into two panels. The left panel shows C code for a function `myInc` that takes an integer `num` and returns `num + 1`. The right panel shows the corresponding assembly code, which consists of three lines: a label `myInc:`, an `add` instruction that adds 1 to the value in register `w0`, and a `ret` instruction to return.

```
2  
3 int myInc(int num) {  
4     return num + 1;  
5 }  
  
1 myInc:  
2     add    w0, w0, #1  
3     ret
```

Source: <https://godbolt.org/>

JIT Compilation

```
java -XX:+PrintCompilation MyTest
72 1 3 java.lang.Object::<init> (1 bytes)
73 2 3 java.lang.StringLatin1::hashCode (42 bytes)
73 4 3 java.lang.String::hashCode (49 bytes)
74 3 3 java.lang.String::isLatin1 (19 bytes)
74 5 3 java.util.ImmutableCollections$SetN::probe (56 bytes)
75 8 3 java.lang.String::coder (15 bytes)
75 6 3 java.lang.Math::floorMod (10 bytes)
75 7 3 java.lang.Math::floorDiv (22 bytes)
77 10 3 java.lang.StringLatin1::equals (36 bytes)
77 11 3 java.util.ImmutableCollections$SetN::hashCode (46 bytes)
78 9 3 java.lang.String::equals (65 bytes)
79 13 3 java.util.ImmutableCollections::emptySet (4 bytes)
80 12 3 java.util.Objects::equals (23 bytes)
80 14 3 java.util.Set::of (4 bytes)
81 15 3 java.util.AbstractCollection::<init> (5 bytes)
81 16 3 java.util.ImmutableCollections$AbstractImmutableCollection::<init> (5 bytes)
81 17 3 java.util.Objects::requireNonNull (14 bytes)
82 18 3 java.util.ImmutableCollections$AbstractImmutableSet::<init> (5 bytes)
82 19 4 java.lang.Object::<init> (1 bytes)
82 20 3 java.util.Set::of (66 bytes)
83 1 3 java.lang.Object::<init> (1 bytes) made not entrant
83 21 1 java.lang.module.ModuleDescriptor::name (5 bytes)
84 22 1 java.lang.module.ModuleReference::descriptor (5 bytes)
89 23 3 java.lang.String::charAt (25 bytes)
89 25 3 java.util.concurrent.ConcurrentHashMap::tabAt (22 bytes)
90 27 3 jdk.internal.misc.Unsafe::getObjectAcquire (7 bytes)
90 26 3 java.util.ImmutableCollections$SetN$SetNIterator::hasNext (13 bytes)
```

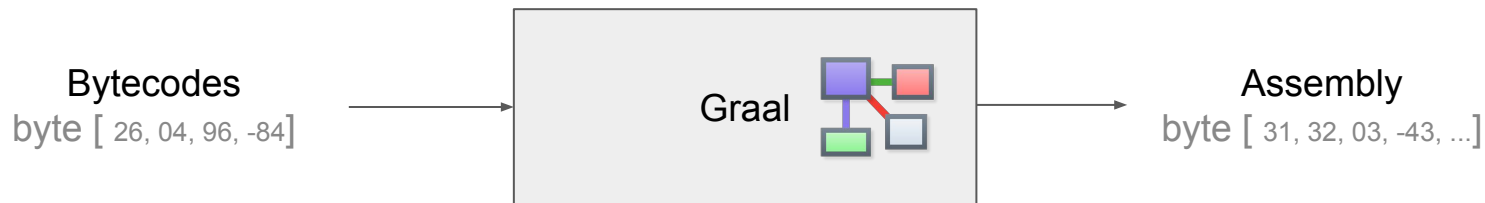
```
$ java -XX:+PrintCompilation MyTest
```

GraalVM™

Graal Compiler



Graal Compiler



```
class Graal {  
    byte[] compile (byte[] method) {  
        ...  
    }  
}
```

Graal Compiler

The screenshot displays an IDE window with the file `HotSpotGraalCompiler.java` open. The code is a Java method `compile` that takes a `StructuredGraph` and returns a `CompilationResult`. The method is annotated with `@` and contains several parameters and a return statement. The debugger is active, showing the `HotSpotGraalCompiler` class in the `Frames` pane. The `Variables` pane shows the state of the method's variables, with `this` and `result` highlighted by red boxes. The `Variables` pane also shows the state of the `graph`, `method`, `entryBCI`, `useProfilingInfo`, `shouldRetainLocalVariables`, `compilationId`, `debug`, and `result` variables.

```
247
248 @
249 public CompilationResult compile(StructuredGraph graph, graph: "StructuredGraph:1{HotSpotMethod<MyTest.myInc(int)>}"
250     ResolvedJavaMethod method, method: "HotSpotMethod<MyTest.myInc(int)>"
251     int entryBCI, entryBCI: -1
252     boolean useProfilingInfo, useProfilingInfo: true
253     boolean shouldRetainLocalVariables, shouldRetainLocalVariables: true
254     CompilationIdentifier compilationId, compilationId: "HotSpotCompilation-183[MyTest.myInc(int)]"
255     DebugContext debug) { debug: DebugContext@2403
256     CompilationResult result = new CompilationResult(compilationId); compilationId: "HotSpotCompilation-183[MyTest.myInc(int)]" result: "org.graalvm.comp
257     return compileHelper(CompilationResultBuilderFactory.Default, result, graph, method, entryBCI, useProfilingInfo, shouldRetainLocalVariables, debug.getOpt
258 }
```

Debug: d05

Debugger Console

Frames

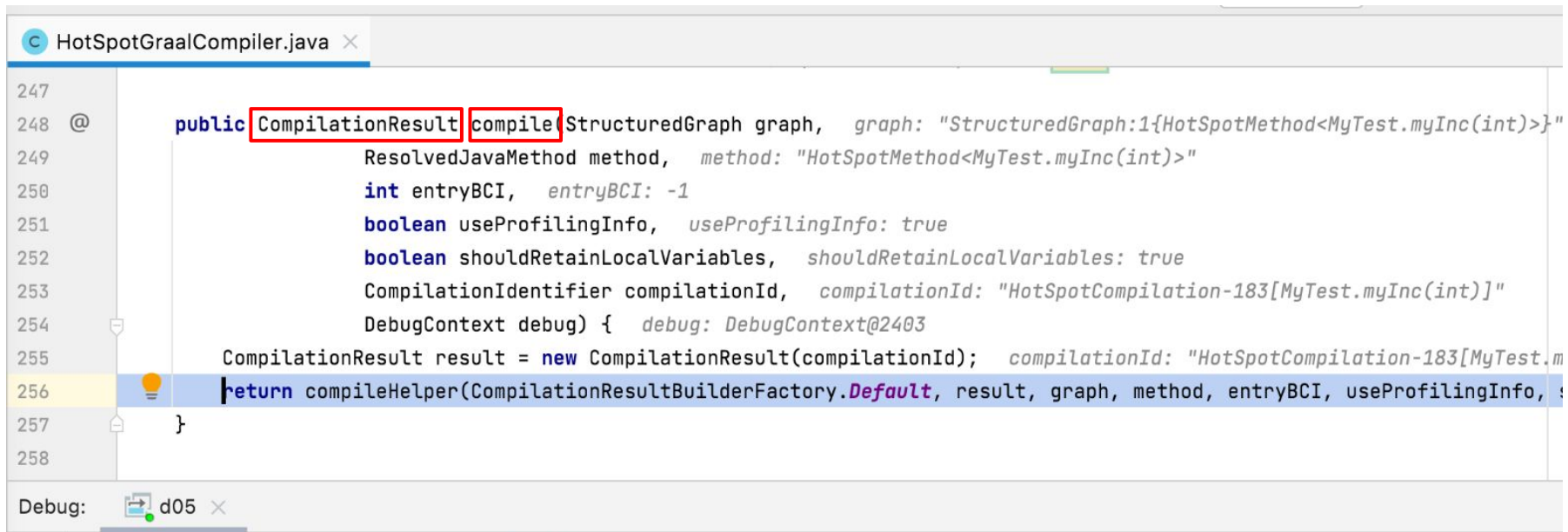
- ✓ "JVMCI Compiler...d0" @607: RUNNING
- compileHelper:238, HotSpotGraalCompiler (org.graalvm.compiler)
- compile:256, HotSpotGraalCompiler (org.graalvm.compiler.hotspot)
- performCompilation:175, CompilationTask\$HotSpotCompilation
- performCompilation:87, CompilationTask\$HotSpotCompilationV
- run:223, CompilationWrapper (org.graalvm.compiler.core)
- runCompilation:362, CompilationTask (org.graalvm.compiler.hotspot)
- compileMethod:154, HotSpotGraalCompiler (org.graalvm.compiler)
- compileMethod:119, HotSpotGraalCompiler (org.graalvm.compiler)
- compileMethod:110, HotSpotGraalCompiler (org.graalvm.compiler)
- compileMethod:847, HotSpotJVMCIRuntime (jdk.vm.ci.hotspot)

Variables

- this = {HotSpotGraalCompiler@2401}
- graph = {StructuredGraph@2381} "StructuredGraph:1{HotSpotMethod<MyTest.myInc(int)>}"
- method = {HotSpotResolvedJavaMethodImpl@2382} "HotSpotMethod<MyTest.myInc(int)>"
- entryBCI = -1
- useProfilingInfo = true
- shouldRetainLocalVariables = true
- compilationId = {HotSpotCompilationIdentifier@2402} "HotSpotCompilation-183[MyTest.myInc(int)]"
- debug = {DebugContext@2403}
- result = {CompilationResult@2390} "org.graalvm.compiler.code.CompilationResult[MyTest.myInc(int)]int"

mx --dbg=*:9090 vm -XX:-BackgroundCompilation -XX:+UnlockDiagnosticVMOptions -XX:CompileCommand="compileonly, MyTest.myInc" -XX:CompileCommand="print, MyTest.myInc"

Graal Compiler



```
HotSpotGraalCompiler.java x
247
248 @ public CompilationResult compile(StructuredGraph graph, graph: "StructuredGraph:1{HotSpotMethod<MyTest.myInc(int)>}"
249     ResolvedJavaMethod method, method: "HotSpotMethod<MyTest.myInc(int)>"
250     int entryBCI, entryBCI: -1
251     boolean useProfilingInfo, useProfilingInfo: true
252     boolean shouldRetainLocalVariables, shouldRetainLocalVariables: true
253     CompilationIdentifier compilationId, compilationId: "HotSpotCompilation-183[MyTest.myInc(int)]"
254     DebugContext debug) { debug: DebugContext@2403
255     CompilationResult result = new CompilationResult(compilationId); compilationId: "HotSpotCompilation-183[MyTest.m
256     return compileHelper(CompilationResultBuilderFactory.Default, result, graph, method, entryBCI, useProfilingInfo, s
257 }
258
Debug: d05 x
```

Graal Compiler

Variables

```
> this = {HotSpotGraalCompiler@2401}
> graph = {StructuredGraph@2381} "StructuredGraph:1{HotSpotMethod<MyTest.myInc(int)>}"
> method = {HotSpotResolvedJavaMethodImpl@2382} "HotSpotMethod<MyTest.myInc(int)>"
  entryBCI = -1
  useProfilingInfo = true
  shouldRetainLocalVariables = true
> compilationId = {HotSpotCompilationIdentifier@2402} "HotSpotCompilation-183[MyTest.myInc(int)]"
> debug = {DebugContext@2403}
> result = {CompilationResult@2390} "org.graalvm.compiler.code.CompilationResult[MyTest.myInc(int)int]"
```

Graal Compiler

Variables

```
> this = {HotSpotGraalCompiler@2401}
> p graph = {StructuredGraph@2381} "StructuredGraph:1{HotSpotMethod<MyTest.myInc(int)>}"
v p method = {HotSpotResolvedJavaMethodImpl@2382} "HotSpotMethod<MyTest.myInc(int)>"
  f metadataHandle = 281472158463232
  > f holder = {HotSpotResolvedObjectTypeImpl@2464} "HotSpotType<LMyTest;, resolved>"
  > f constantPool = {HotSpotConstantPool@2465} "HotSpotConstantPool<MyTest>"
  > f signature = {HotSpotSignature@2466} "HotSpotSignature<(I)I>"
  > f methodData = {HotSpotMethodData@2467} "Raw method data for MyTest.myInc(int):\n\nof_decompile... View
  > f code = {byte[4]@3248} [26, 4, 96, -84]
  f toJavaCache = null
  > f nameCache = "myInc"
```

Graal Compiler

Variables

+

>

≡

this = {HotSpotGraalCompiler@2401}

>

p

graph = {StructuredGraph@2381} "StructuredGraph:1{HotSpotMethod<MyTest>@2380} [26, 4, 96, -84]"

∨

p

method = {HotSpotResolvedJavaMethodImpl@2382} "HotSpotMethod<MyTest>@2380"

f

metadataHandle = 281472158463232

>

f

holder = {HotSpotResolvedObjectTypeInfo@2464} "HotSpotType<LMyTest; resolved>"

>

f

constantPool = {HotSpotConstantPool@2465} "HotSpotConstantPool<MyTest>"

>

f

signature = {HotSpotSignature@2466} "HotSpotSignature<(I)I>"

>

f

methodData = {HotSpotMethodData@2467} "Raw method data for MyTest.myInc(int):\nnof_decompile..." [View](#)

>

f

code = {byte[4]@3248} [26, 4, 96, -84]

f

toJavaCache = null

>

f

nameCache = "myInc"

iload_0	1a
iconst_1	04
iadd	60
ireturn	ac

Graal Compiler

Variables

- > **p** debug = {DebugContext@2403}
- ✓ **result** = {CompilationResult@2390} "org.graalvm.compiler.code.CompilationResult[MyTest.myInc(int)int]"
 - f** closed = true
 - f** entryBCI = -1
 - > **f** dataSection = {DataSection@3393} "org.graalvm.compiler.code.DataSection@478043324"
 - > **f** infopoints = {ArrayList@3394} size = 2
 - f** sourceMapping = {ArrayList@3395} size = 0
 - f** dataPatches = {ArrayList@3396} size = 0
 - f** exceptionHandlers = {ArrayList@3397} size = 0
 - > **f** marks = {ArrayList@3398} size = 5
 - f** totalFrameSize = 32
 - f** maxInterpreterFrameSize = 0
 - f** customStackArea = null
 - f** name = null
 - > **f** compilationId = {HotSpotCompilationIdentifier@2402} "HotSpotCompilation-183[MyTest.myInc(int)]"
 - > **f** targetCode = {byte[232]@3399} [31, 32, 3, -43, -24, -1, -121, -110, -56, -1, -65, -14, -1, 107, 40, -8, -1, -125, 0, -47, -3, ... View
 - f** targetCodeSize = 92
 - > **f** annotations = {ArrayList@3400} size = 5

Graal Compiler

Variables



> **this** = {HotSpotGraalCompiler@2401}



▽ **graph** = {StructuredGraph@2381} "StructuredGraph:1{HotSpotMethod<MyTest.myInc(int)>}"

nodesSize = 6



▽ **nodes** = {Node[32]@3373}

Not showing null elements



> **0** = {StartNode@3363} "0|StartNode"



> **1** = {ParameterNode@3416} "1|Parameter(0)"



> **3** = {ConstantNode@3417} "3|Constant(1, i32)"



> **4** = {AddNode@3418} "4|+"



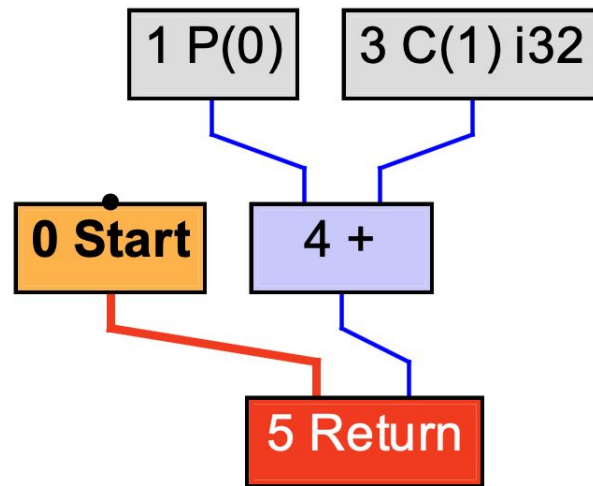
> **5** = {ReturnNode@3408} "5|Return"

Graal Compiler

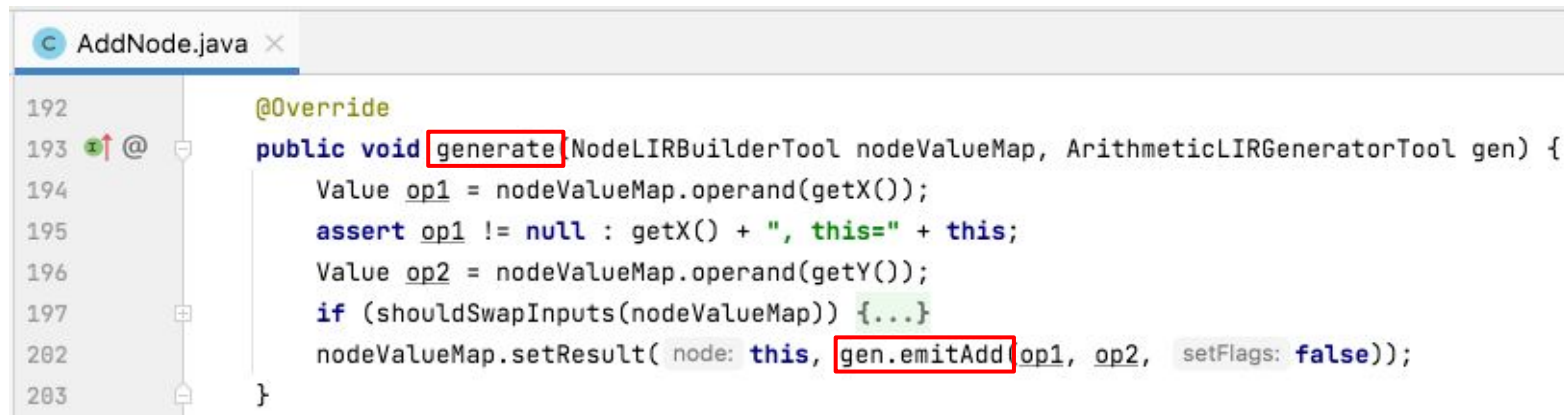
Variables

```
> this = {HotSpotGraalCompiler@2401}
p graph = {StructuredGraph@2381} "StructuredGraph:1{HotSpotMethod<MyTest.myInc(int)>}"
  nodesSize = 6

f nodes = {Node[32]@3373}
  Not showing null elements
  > 0 = {StartNode@3363} "0|StartNode"
  > 1 = {ParameterNode@3416} "1|Parameter(0)"
  > 3 = {ConstantNode@3417} "3|Constant(1, i32)"
  > 4 = {AddNode@3418} "4|+"
  > 5 = {ReturnNode@3408} "5|Return"
```



Emit Code



```
192      @Override
193      public void generate(NodeLIRBuilderTool nodeValueMap, ArithmeticLIRGeneratorTool gen) {
194          Value op1 = nodeValueMap.operand(getX());
195          assert op1 != null : getX() + ", this=" + this;
196          Value op2 = nodeValueMap.operand(getY());
197          if (shouldSwapInputs(nodeValueMap)) {...}
202          nodeValueMap.setResult( node: this, gen.emitAdd(op1, op2, setFlags: false));
203      }
```

Emit Code

```
AddNode.java x
192      @Override
193      public void generate(NodeLIRBuilderTool nodeValueMap, ArithmeticLIRGeneratorTool gen) {
194          Value op1 = nodeValueMap.operand(getX());
195          assert op1 != null : getX() + ", this=" + this;
196          Value op2 = nodeValueMap.operand(getY());
197          if (shouldSwapInputs(nodeValueMap)) {...}
202          nodeValueMap.setResult( node: this, gen.emitAdd(op1, op2, setFlags: false));
203      }
```

```
ArithmeticLIRGenerator.java x
56
57      protected abstract Variable emitAdd(LIRKind resultKind, Value a, Value b, boolean setFlags);
58
59      Choose Implementation of emitAdd (2 methods found)
60      AArch64ArithmeticLIRGenerator (org.graalvm.compiler.core.aarch64) org.graalvm.compiler.core.aarch64
61      AMD64ArithmeticLIRGenerator (org.graalvm.compiler.core.amd64) org.graalvm.compiler.core.amd64
```

Emit Code

```
AddNode.java x
192      @Override
193      public void generate(NodeLIRBuilderTool nodeValueMap, ArithmeticLIRGeneratorTool gen) {
194          Value op1 = nodeValueMap.operand(getX());
195          assert op1 != null : getX() + ", this=" + this;
196          Value op2 = nodeValueMap.operand(getY());
197          if (shouldSwapInputs(nodeValueMap)) {...}

AArch64ArithmeticLIRGenerator.java x
97      @Override
98      protected Variable emitAdd(LIRKind resultKind, Value a, Value b, boolean setFlags) {
99          if (isNumericInteger(a.getPlatformKind())) {
100              AArch64ArithmeticOp op = setFlags ? AArch64ArithmeticOp.ADDS : AArch64ArithmeticOp.ADD;
101              return emitBinary(resultKind, op, commutative: true, a, b);
102          } else {
103              assert !setFlags : "Cannot set flags on floating point arithmetic";
104              return emitBinary(resultKind, AArch64ArithmeticOp.FADD, commutative: true, a, b);
105          }
106      }
```

Code Generation

```
MyTest.myInc(I)I [0x0000ffffdd2d6a80, 0x0000ffffdd2d6ae0] 96 bytes
[Entry Point]
[Verified Entry Point]
[Constants]
# {method} {0x0000ffff7ce2a7d8} 'myInc' '(I)I' in 'MyTest'
# parm0:    c_rarg1    = int
#           [sp+0x20] (sp of caller)
0x0000ffffdd2d6a80: nop
0x0000ffffdd2d6a84: mov     x8, #0xffffffffffffc000    // #-16384
0x0000ffffdd2d6a88: movk    x8, #0xfffe, lsl #16
0x0000ffffdd2d6a8c: str     xzr, [sp,x8]
0x0000ffffdd2d6a90: sub     sp, sp, #0x20
0x0000ffffdd2d6a94: stp     x29, x30, [sp,#16]
0x0000ffffdd2d6a98: add     w0, w1, #0x1    W0 = W1 + 1
0x0000ffffdd2d6a9c: ldp     x29, x30, [sp,#16]
0x0000ffffdd2d6aa0: add     sp, sp, #0x20
0x0000ffffdd2d6aa4: ldr     x8, [x28,#264]
0x0000ffffdd2d6aa8: ldr     wzr, [x8]      ; {poll_return}
0x0000ffffdd2d6aac: ret
```

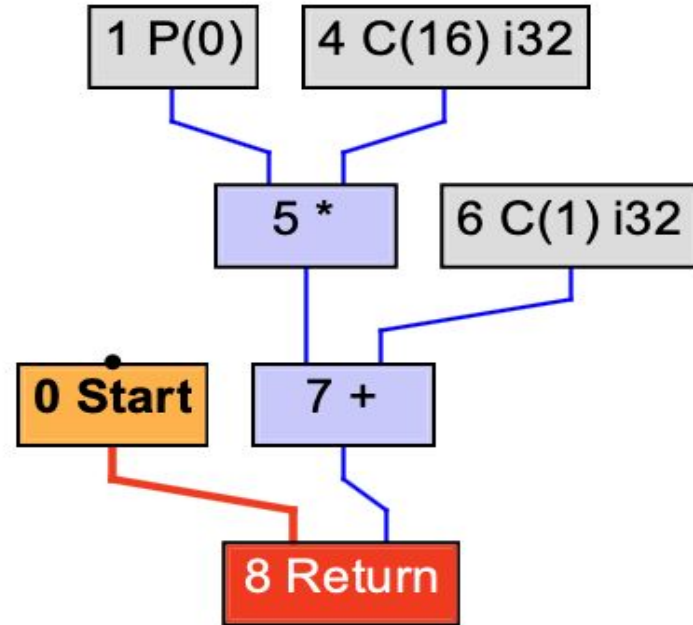
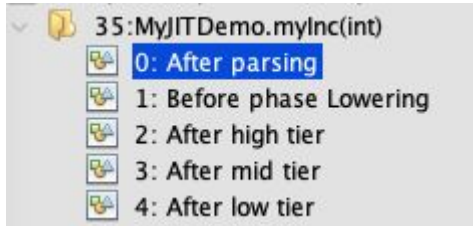
```
$GRAAL_HOME/bin/java -XX:+UnlockExperimentalVMOptions -XX:+EnableJVMCI -XX:+UseJVMCICompiler \
-XX:-BackgroundCompilation -XX:CompileCommand=print,MyTest.myInc' MyTest
```

Compiler Optimisations

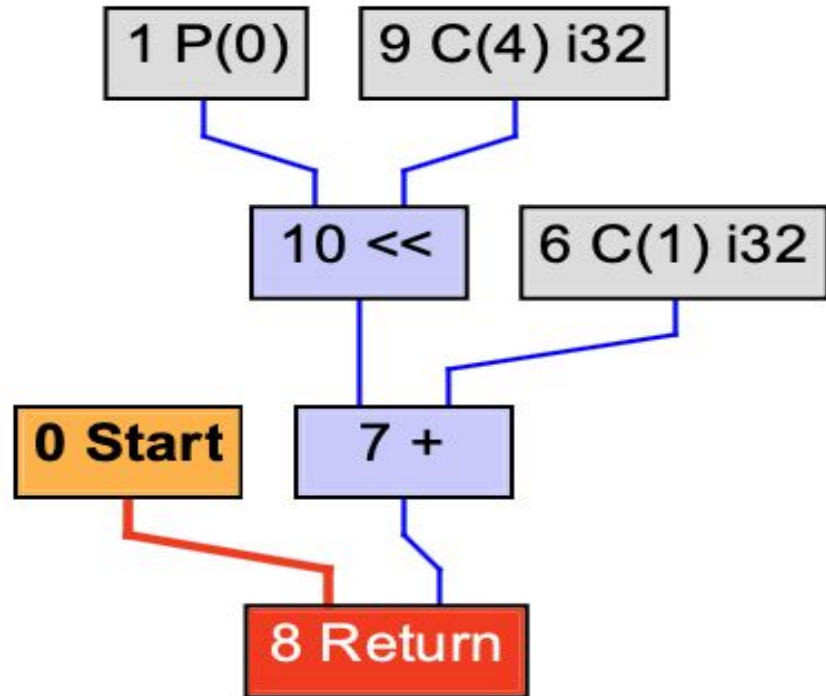
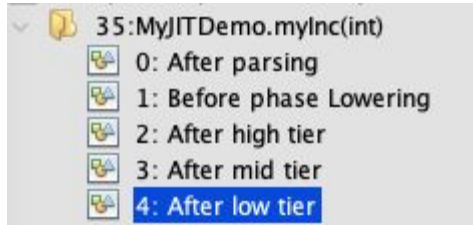
```
public class MyJITDemo {  
    private static int myMult(final int a) {  
        return a * 16;  
    }  
  
    public static int myInc(int a) {  
        a = myMult(a);  
        return a + 1;  
    }  
  
    public static void main(String[] args) {  
        for(int i = 0; i < 1_000_000; i++)  
            MyJITDemo.myInc(i);  
    }  
}
```

```
$GRAAL_HOME/bin/java -server -XX:+UnlockExperimentalVMOptions -XX:+EnableJVMCI -XX:+UseJVMCICompiler -Dgraal.Dump=: \  
-XX:-BackgroundCompilation '-XX:CompileCommand=compileonly, MyJITDemo.myInc' '-XX:CompileCommand=print, MyJITDemo.myInc' MyJITDemo
```


























Graal Compiler: Before Compilation




























Graal Compiler: After Compilation



Compilation Pipeline (High Tier)


























-  0: After phase GraphBuilder
-  1: After phase PhaseSuite
-  2: After phase DeadCodeElimination
-  3: After parsing
-  4: After phase Canonicalizer
-  5: After phase Inlining
-  6: After phase DeadCodeElimination
-  7: After phase DisableOverflowCountedLoops
-  8: After phase ConvertDeoptimizeToGuard
-  9: After phase IncrementalCanonicalizer
-  10: After phase IterativeConditionalElimination
-  11: After phase LoopFullUnroll
-  12: After phase LoopPeeling
-  13: After phase IncrementalCanonicalizer
-  14: After phase LoopUnswitching
-  15: After phase IncrementalCanonicalizer
-  16: After phase BoxNodeIdentity
-  17: After phase PartialEscape
-  18: After phase ReadElimination
-  19: After phase BoxNodeOptimization
-  20: After phase IncrementalCanonicalizer
-  21: Before phase Lowering
-  22: After phase Lowering
-  23: After phase HighTier
-  24: After high tier

Compilation Pipeline (Mid Tier)













-  0: After phase GraphBuilder
-  1: After phase PhaseSuite
-  2: After phase DeadCodeElimination
-  3: After parsing
-  4: After phase Canonicalizer
-  5: After phase Inlining
-  6: After phase DeadCodeElimination
-  7: After phase DisableOverflowCountedLoops
-  8: After phase ConvertDeoptimizeToGuard
-  9: After phase IncrementalCanonicalizer
-  10: After phase IterativeConditionalElimination
-  11: After phase LoopFullUnroll
-  12: After phase LoopPeeling
-  13: After phase IncrementalCanonicalizer
-  14: After phase LoopUnswitching
-  15: After phase IncrementalCanonicalizer
-  16: After phase BoxNodeIdentity
-  17: After phase PartialEscape
-  18: After phase ReadElimination
-  19: After phase BoxNodeOptimization
-  20: After phase IncrementalCanonicalizer
-  21: Before phase Lowering
-  22: After phase Lowering
-  23: After phase HighTier
-  24: After high tier

-  24: After high tier
-  25: After phase LockElimination
-  26: After phase FloatingRead
-  27: After phase IncrementalCanonicalizer
-  28: After phase IterativeConditionalElimination
-  29: After phase LoopSafePointElimination
-  30: After phase SpeculativeGuardMovement
-  31: After phase IncrementalCanonicalizer
-  32: After phase GuardLowering
-  33: After phase RemoveValueProxy
-  34: After phase IncrementalCanonicalizer
-  35: After phase LoopSafePointInsertion
-  36: After phase Lowering
-  37: After phase IterativeConditionalElimination
-  38: After phase OptimizeDiv
-  39: After phase FrameStateAssignment
-  40: After phase LoopPartialUnroll
-  41: After phase Reassociation
-  42: After phase DeoptimizationGrouping
-  43: After phase Canonicalizer
-  44: After phase WriteBarrierAddition
-  45: After phase MidTier
-  46: After mid tier

Compilation Pipeline (Low Tier)

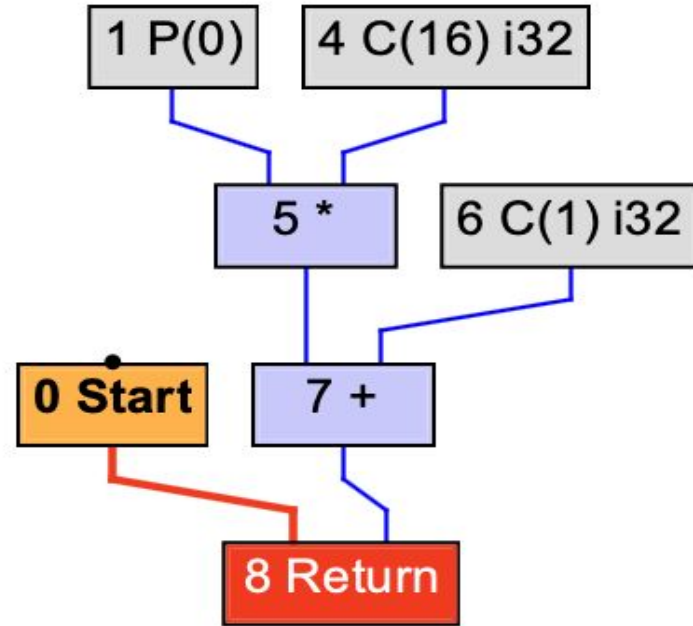
-  0: After phase GraphBuilder
-  1: After phase PhaseSuite
-  2: After phase DeadCodeElimination
-  3: After parsing
-  4: After phase Canonicalizer
-  5: After phase Inlining
-  6: After phase DeadCodeElimination
-  7: After phase DisableOverflowCountedLoops
-  8: After phase ConvertDeoptimizeToGuard
-  9: After phase IncrementalCanonicalizer
-  10: After phase IterativeConditionalElimination
-  11: After phase LoopFullUnroll
-  12: After phase LoopPeeling
-  13: After phase IncrementalCanonicalizer
-  14: After phase LoopUnswitching
-  15: After phase IncrementalCanonicalizer
-  16: After phase BoxNodeIdentity
-  17: After phase PartialEscape
-  18: After phase ReadElimination
-  19: After phase BoxNodeOptimization
-  20: After phase IncrementalCanonicalizer
-  21: Before phase Lowering
-  22: After phase Lowering
-  23: After phase HighTier
-  24: After high tier

-  24: After high tier
-  25: After phase LockElimination
-  26: After phase FloatingRead
-  27: After phase IncrementalCanonicalizer
-  28: After phase IterativeConditionalElimination
-  29: After phase LoopSafePointElimination
-  30: After phase SpeculativeGuardMovement
-  31: After phase IncrementalCanonicalizer
-  32: After phase GuardLowering
-  33: After phase RemoveValueProxy
-  34: After phase IncrementalCanonicalizer
-  35: After phase LoopSafePointInsertion
-  36: After phase Lowering
-  37: After phase IterativeConditionalElimination
-  38: After phase OptimizeDiv
-  39: After phase FrameStateAssignment
-  40: After phase LoopPartialUnroll
-  41: After phase Reassociation
-  42: After phase DeoptimizationGrouping
-  43: After phase Canonicalizer
-  44: After phase WriteBarrierAddition
-  45: After phase MidTier
-  46: After mid tier

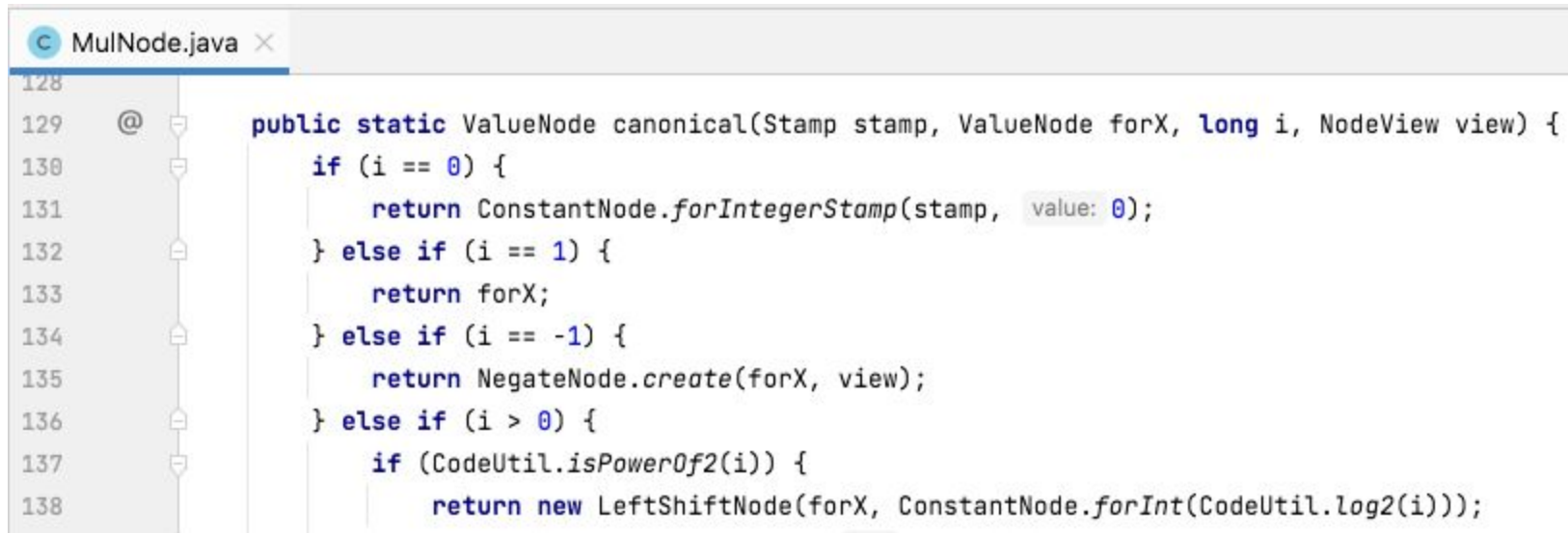
-  46: After mid tier
-  47: After phase Lowering
-  48: After phase ExpandLogic
-  49: After phase FixReads
-  50: After phase UseTrappingNullChecks
-  51: After phase AddressLowering
-  52: After phase Canonicalizer
-  53: After phase DeadCodeElimination
-  54: After phase PropagateDeoptimizeProbability
-  55: After phase Schedule
-  56: After phase LowTier
-  57: After low tier

Optimising a graph

- `canonical()`
- `generate()`



Simplify multiplication



The screenshot shows a code editor with a tab labeled 'MulNode.java'. The code is in Java and defines a static method 'canonical' that takes a 'Stamp', a 'ValueNode' (forX), a 'long' (i), and a 'NodeView' (view) as parameters. The method implements a series of conditional checks to simplify multiplication based on the value of 'i'. Line numbers 128 through 138 are visible on the left margin. A vertical toolbar with icons for undo, redo, and other editing functions is located between the line numbers and the code.

```
128
129  @ public static ValueNode canonical(Stamp stamp, ValueNode forX, long i, NodeView view) {
130      if (i == 0) {
131          return ConstantNode.forIntegerStamp(stamp, value: 0);
132      } else if (i == 1) {
133          return forX;
134      } else if (i == -1) {
135          return NegateNode.create(forX, view);
136      } else if (i > 0) {
137          if (CodeUtil.isPowerOf2(i)) {
138              return new LeftShiftNode(forX, ConstantNode.forInt(CodeUtil.log2(i)));
```

Optimised Assembly

```
MyJITDemo.myInc(I)I [0x0000ffffdd2d6d80, 0x0000ffffdd2d6de0] 96 bytes
[Entry Point]
[Verified Entry Point]
[Constants]
# {method} {0x0000ffff7ce2a898} 'myInc' '(I)I' in 'MyJITDemo'
# parm0:    c_rarg1    = int
#           [sp+0x20] (sp of caller)
0x0000ffffdd2d6d80: nop
0x0000ffffdd2d6d84: mov     x8, #0xffffffffffffc000    // #-16384
0x0000ffffdd2d6d88: movk    x8, #0xfffe, lsl #16
0x0000ffffdd2d6d8c: str     xzr, [sp,x8]
0x0000ffffdd2d6d90: sub     sp, sp, #0x20
0x0000ffffdd2d6d94: stp     x29, x30, [sp,#16]
0x0000ffffdd2d6d98: orr     w0, wzr, #0x1
0x0000ffffdd2d6d9c: add     w0, w0, w1, lsl #4
                                ; -
                                {rethrow=0 return_oop=0}
                                MyJITDemo.myInc7 (line 10)
                                ; -

0x0000ffffdd2d6da0: ldp     x29, x30, [sp,#16]
0x0000ffffdd2d6da4: add     sp, sp, #0x20
0x0000ffffdd2d6da8: ldr     x8, [x28,#264]
0x0000ffffdd2d6dac: ldr     wzr, [x8] ; {poll_return}
0x0000ffffdd2d6db0: ret
```


Platform specific instructions

AArch64NodeMatchRules.java

```
316 @MatchRule("(Add=op x (LeftShift=lshift (SignExtend=ext y) Constant))")
317 public ComplexMatchResult mergeSignExtendByShiftIntoAddSub(BinaryNode op, LeftShiftNode lshift,
318                                                             ValueNode ext, ValueNode x, ValueNode y) {
319     assert isNumericInteger(lshift);
320     int shiftAmt = getClampedShiftAmt(lshift);
321     if (!isSupportedExtendedAddSubShift((IntegerConvertNode<?, ?>) ext, shiftAmt)) {...}
322     ExtendType extType;
323     if (ext instanceof SignExtendNode) {...} else {...}
324     return emitExtendedAddSubShift(op, x, y, extType, shiftAmt);
325 }
```


Summary

- Graal as a Just-in-Time (JIT) compiler
- Graph representation of a Java program
- Compiler Optimisations
 - Platform specific
 - Platform agnostic
- Generate assembly instructions



<https://github.com/SwapnilGaikwad/devoxx22>

Things for another talk ...

- Instruction Scheduling
- Register Allocation
- Graal Intrinsics
- Graal Native Image
- Graal for running other languages
- Other JVM Components

Thank You!

Questions?

Resources

- Graal Repo: <https://github.com/oracle/graal>
- Graal Publications & Videos: <https://github.com/oracle/graal/blob/master/docs/Publications.md>
- Graal Compiler Docs: <https://github.com/oracle/graal/tree/master/compiler/docs>
- Top 10 Things To Do With GraalVM:
<https://medium.com/graalvm/graalvm-ten-things-12d9111f307d>
- IdealGraphVisualizer:
<https://github.com/oracle/graal/blob/master/docs/tools/ideal-graph-visualizer.md>

<https://github.com/SwapnilGaikwad/devoxx22>

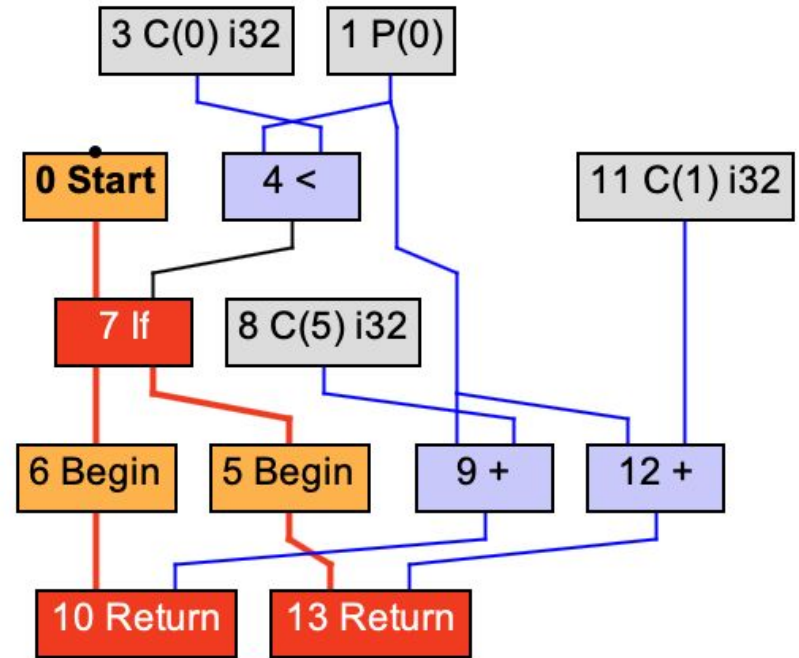


Challenges

- Benchmarking
 - Prove the optimisation improves performance
 - Analyse behaviour on different platforms
- Prototyping
 - Test a given sequence of instructions
 - Identify suitable instructions

Graal Compiler

```
public class MyTest {  
    public static int myInc(int a) {  
        if(a < 0){  
            return a + 5;  
        }  
        return a + 1;  
    }  
  
    public static void main(String[] args)  
    {  
        int i = -3;  
        while(true) {  
            MyTest.myInc(i++);  
            if(i > 100)  
                i = -10;  
        }  
    }  
}
```



Optimisation Phase

```
OptimizeDivPhase.java x
48 public class OptimizeDivPhase extends Phase {
49
50     @Override
51     protected void run(StructuredGraph graph) {
52         for (IntegerDivRemNode rem : graph.getNodes(IntegerDivRemNode.TYPE)) {
53             if (rem instanceof SignedRemNode && divByNonZeroConstant(rem)) {
54                 optimizeRem(rem);
55             }
56         }
57         for (IntegerDivRemNode div : graph.getNodes(IntegerDivRemNode.TYPE)) {...}
62     }
```

Optimisation Phase

```
OptimizeDivPhase.java x
73  @ protected final void optimizeRem(IntegerDivRemNode rem) {
74      assert rem.getOp() == IntegerDivRemNode.Op.REM;
75      // Java spec 15.17.3.: (a/b)*b+(a%b) == a
76      // so a%b == a-(a/b)*b
77      StructuredGraph graph = rem.graph();
78      ValueNode div = findDivForRem(rem);
79      ValueNode mul = BinaryArithmeticNode.mul(graph, div, rem.getY(), NodeView.DEFAULT);
80      ValueNode result = BinaryArithmeticNode.sub(graph, rem.getX(), mul, NodeView.DEFAULT);
81      graph.replaceFixedWithFloating(rem, result);
82  }
```