JOHANNES KEPLER UNIVERSITY LINZ | JKU

Graal : going deeper in the JVM using Java

JVM Language Summit 2012

Gilles Duboscq

JOHANNES KEPLER UNIVERSITY LINZ | JKU

Gilles Duboscq

- Working at Johannes Kepler University Linz, Austria
- Working on the Graal project since april 2011

Graal

- Java APIs on top of Hotspot
- Compiler

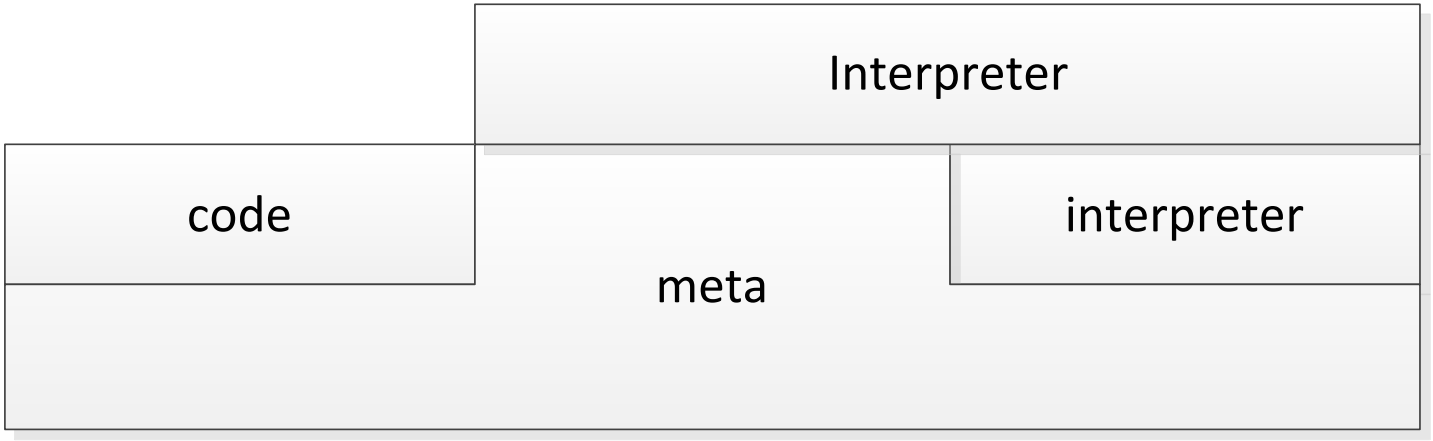
Graal APIs

- meta
  - Lazy Reflection (read only)
  - Profiling
- code
  - native code installation and execution
  - Hot method callback
- interpreter
  - Write extension to Lazy Reflection
  - Method execution callback

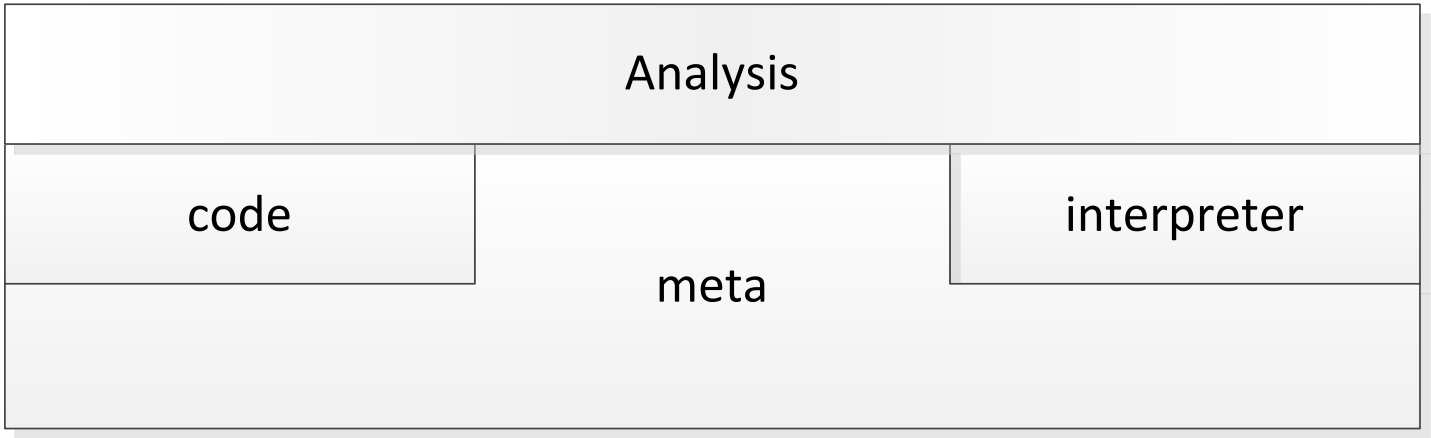
Graal APIs

Graal APIs

Graal APIs



Graal APIs



Analysis

```
public void processMethod(CallGraphNode node) {
    StructuredGraph graph = new StructuredGraph(node.method);
    new GraphBuilderPhase(metaAccess, config, OptimisticOptimizations.NONE).apply(graph);
    new CanonicalizerPhase(null, metaAccess, null).apply(graph);
    new IterativeCheckCastEliminationPhase(null, metaAccess, null).apply(graph);

    for (MethodCallTargetNode callTarget : graph.getNodes(MethodCallTargetNode.class)) {
        node.edges.add(getNode(callTarget.targetMethod()));
    }
}
```

Bytecode Interpreter

```
private void putFieldVirtual(InterpreterFrame frame, ResolvedJavaField field) {
    switch (field.kind()) {
        ...
        case Object:
            runtimeInterface.setFieldObject(frame.popObject(), nullCheck(frame.popObject()), field);
            break;
    }
}
```

Bytecode Interpreter

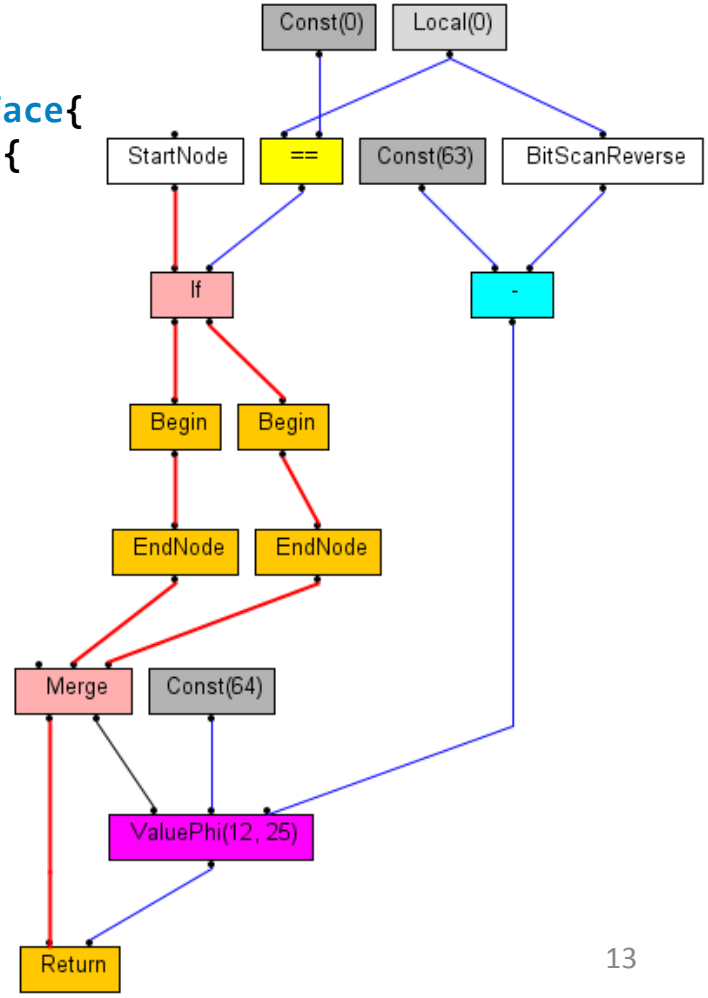
```
private void putFieldVirtual(InterpreterFrame frame, ResolvedJavaField field) {
    switch (field.kind()) {
        ...
        case Object:
            Object value = frame.popObject();
            if (field.getAnnotation(NonNull.class) != null && value == null) {
                throw new RuntimeException("Storing null into a @NonNull field");
            }
            runtimeInterface.setFieldObject(value, nullCheck(frame.popObject()), field);
            break;
    }
}
```

Graal IR

- SSA
- Named edges
- Easy iteration

Graal IR - Snippets

```
@ClassSubstitution(Long.class)
public class LongSnippets implements SnippetsInterface{
    public static int numberOfLeadingZeros(long i) {
        if (i == 0) {
            return 64;
        }
        return 63 - BitScanReverseNode.scan(i);
    }
}
```



Graal IR - Snippets

```
public class BitScanForwardNode extends FloatingNode implements LIRGenLowerable {
    @Input private ValueNode value;
    public BitScanForwardNode(ValueNode value) {
        super(StampFactory.forInteger(Kind.Int, 0, value.kind().bits()));
        this.value = value;
    }
    @NodeIntrinsic
    public static int scan(long v) {
        throw new UnsupportedOperationException();
    }
    @Override
    public void generate(LIRGenerator gen) {
        Variable result = gen.newVariable(Kind.Int);
        gen.append(new AMD64BitScanOp(IntrinsicOpcode.BSF, result,
gen.operand(value)));
        gen.setResult(this, result);
    }
}
```

Graal IR - Snippets

```
public class BitScanForwardNode extends FloatingNode implements LIRGenLowerable {
    @Input private ValueNode value;
    public BitScanForwardNode(ValueNode value) {
        super(StampFactory.forInteger(Kind.Int, 0, value.kind().bits()));
        this.value = value;
    }
    @NodeIntrinsic
    public static int scan(long v) {
        throw new UnsupportedOperationException();
    }
    @Override
    public void generate(LIRGenerator gen) {
        Variable result = gen.newVariable(Kind.Int);
        gen.append(new AMD64BitScanOp(IntrinsicOpcode.BSF, result,
gen.operand(value)));
        gen.setResult(this, result);
    }
}
```

Graal IR - Snippets

```
public class BitScanForwardNode extends FloatingNode implements LIRGenLowerable {
    @Input private ValueNode value;
    public BitScanForwardNode(ValueNode value) {
        super(StampFactory.forInteger(Kind.Int, 0, value.kind().bits()));
        this.value = value;
    }
    @NodeIntrinsic
    public static int scan(long v) {
        throw new UnsupportedOperationException();
    }
    @Override
    public void generate(LIRGenerator gen) {
        Variable result = gen.newVariable(Kind.Int);
        gen.append(new AMD64BitScanOp(IntrinsicOpcode.BSF, result,
gen.operand(value)));
        gen.setResult(this, result);
    }
}
```

Snippets - arraycopy

```
@Snippet
public static void arraycopy(char[] src, int srcPos, char[] dest, int destPos,
int length) {
    if (src == null || dest == null) {
        throw new NullPointerException();
    }
    if (srcPos < 0 || destPos < 0 || length < 0 || srcPos + length > src.length
|| destPos + length > dest.length) {
        throw new IndexOutOfBoundsException();
    }
    vectorizedCopy(src, srcPos, dest, destPos, length, Kind.Char);
}
```

Snippets - arraycopy

```
@Snippet
public static void vectorizedCopy(Object src, int srcPos, Object dest, int destPos, int length, Kind baseKind) {
    int header = arrayBaseOffset(baseKind);
    int elementSize = arrayIndexScale(baseKind);
    long byteLength = length * elementSize;
    long nonVectorBytes = byteLength % VECTOR_SIZE;
    long srcOffset = srcPos * elementSize;
    long destOffset = destPos * elementSize;
    if (src == dest && srcPos < destPos) { // aliased case
        for (long i = byteLength - elementSize; i >= byteLength - nonVectorBytes; i -= elementSize) {
            UnsafeStoreNode.store(dest, header, i + destOffset, UnsafeLoadNode.load(src, header, i + srcOffset,
baseKind), baseKind);
        }
        long vectorLength = byteLength - nonVectorBytes;
        for (long i = vectorLength - VECTOR_SIZE; i >= 0; i -= VECTOR_SIZE) {
            Long a = UnsafeLoadNode.load(src, header, i + srcOffset, VECTOR_KIND);
            UnsafeStoreNode.store(dest, header, i + destOffset, a.longValue(), VECTOR_KIND);
        }
    } else {
        for (long i = 0; i < nonVectorBytes; i += elementSize) {
            UnsafeStoreNode.store(dest, header, i + destOffset, UnsafeLoadNode.load(src, header, i + srcOffset,
baseKind), baseKind);
        }
        for (long i = nonVectorBytes; i < byteLength; i += VECTOR_SIZE) {
            Long a = UnsafeLoadNode.load(src, header, i + srcOffset, VECTOR_KIND);
            UnsafeStoreNode.store(dest, header, i + destOffset, a.longValue(), VECTOR_KIND);
        }
    }
}
```

Graal IR – Low/Tupple IR

```
@Opcode("MOVE")
public static class MoveToRegOp extends AMD64LIRInstruction implements MoveOp {
    @Def({REG, HINT}) protected Value result;
    @Use({REG, STACK, CONST}) protected Value input;
    public MoveToRegOp(Value result, Value input) {
        this.result = result;
        this.input = input;
    }
    @Override
    public void emitCode(TargetMethodAssembler tasm, AMD64MacroAssembler masm) {
       move(tasm, masm, getResult(), getInput());
    }
}
```

Graal – Type analysis

- Kind
- Stamp
  - IntegerStamp = [min, max] + mask
  - FloatStamp = [min, max] + nonNaN flag
  - ObjectStamp = Java Type + nullness flags

Graal - Phase

```
public class NonNullOptPhase extends Phase {
    @Override
    protected void run(StructuredGraph graph) {
        for (LoadFieldNode load : graph.getNodes(LoadFieldNode.class)) {
            if (load.kind().isObject()) {
                if (load.field().getAnnotation(NonNull.class) != null) {
                    setNonNullStamp(load, load.field().type());
                }
            }
        }
        for (MethodCallTargetNode callTarget : graph.getNodes(MethodCallTargetNode.class)) {
            if (callTarget.returnKind().isObject()) {
                if (callTarget.targetMethod().getAnnotation(NonNull.class) != null) {
                    setNonNullStamp(callTarget.invoke().node(), callTarget.returnType());
                }
            }
        }
    }
    private static void setNonNullStamp(ValueNode node, JavaType type) {
        if (type instanceof ResolvedJavaType) {
            node.setStamp(StampFactory.declaredNonNull((ResolvedJavaType) type));
        } else {
            node.setStamp(StampFactory.objectNonNull());
        }
    }
}
```

Compilation Process

Compilation Process

- "Simple" graph building from bytecode
  - Transformations :
    - From linear bytecode sequence to CFG
    - From stack-based representation to SSA
  - Simple nodes (close to Java bytecodes)
    - LoadFieldNode, InstanceOfNode…
  - No inlining

Compilation Process

- High level optimizations
  - Inlining
  - Escape Analysis
  - Intrinsics
  - Loop peeling
  - Full unrolling
  - Control Flow sensitive canonicalization

Compilation Process

- Lowering
  - From a high level concept to its implementation
  - Happens for multiple reasons
    - Runtime specificities
    - Target architecture specificities
  - Progressive lowering can offer new optimization opportunities
    - Canonicalization
    - Global Value Numbering

Compilation Process

- Intermediate optimizations
  - Floating reads
  - Read elimination
  - Tail duplication
  - Control Flow sensitive canonicalization

Compilation Process

- Scheduling
  - Performs code motion optimizations “naturally”
    - “Hoists” loop invariant code out of loops

Compilation Process

- Architecture specific lowering
  - De-SSA
  - Lowering to almost IR Node ⇔ x86 instruction

Compilation Process

- Back-end
  - Register allocation
  - Code emitting

Compilation Process

- Small recurrent Phases
  - Data Flow Canonicalization
  - Control Flow Simplification
  - Global Value Numbering
  - Type propagation

Compilation Process

- Global/Shared optimizations
  - Dead code elimination on branch removal
  - Global Value Numbering on insertion

How do you get Graal?

- `hg clone http://hg.openjdk.java.net/graal/graal`
- `cd graal`
- `echo "JAVA_HOME=..." > mx/env`
- `mx build`
- `mx ideinit`

- Tested on Linux, Windows and OS X (64 bits)
  - Python 2.7
  - a bootstrap JDK7
  - Cygwin and Microsoft Windows SDK on Windows

Questions?