Flow-sensitive rewritings and Inliner improvements for the Graal JIT compiler

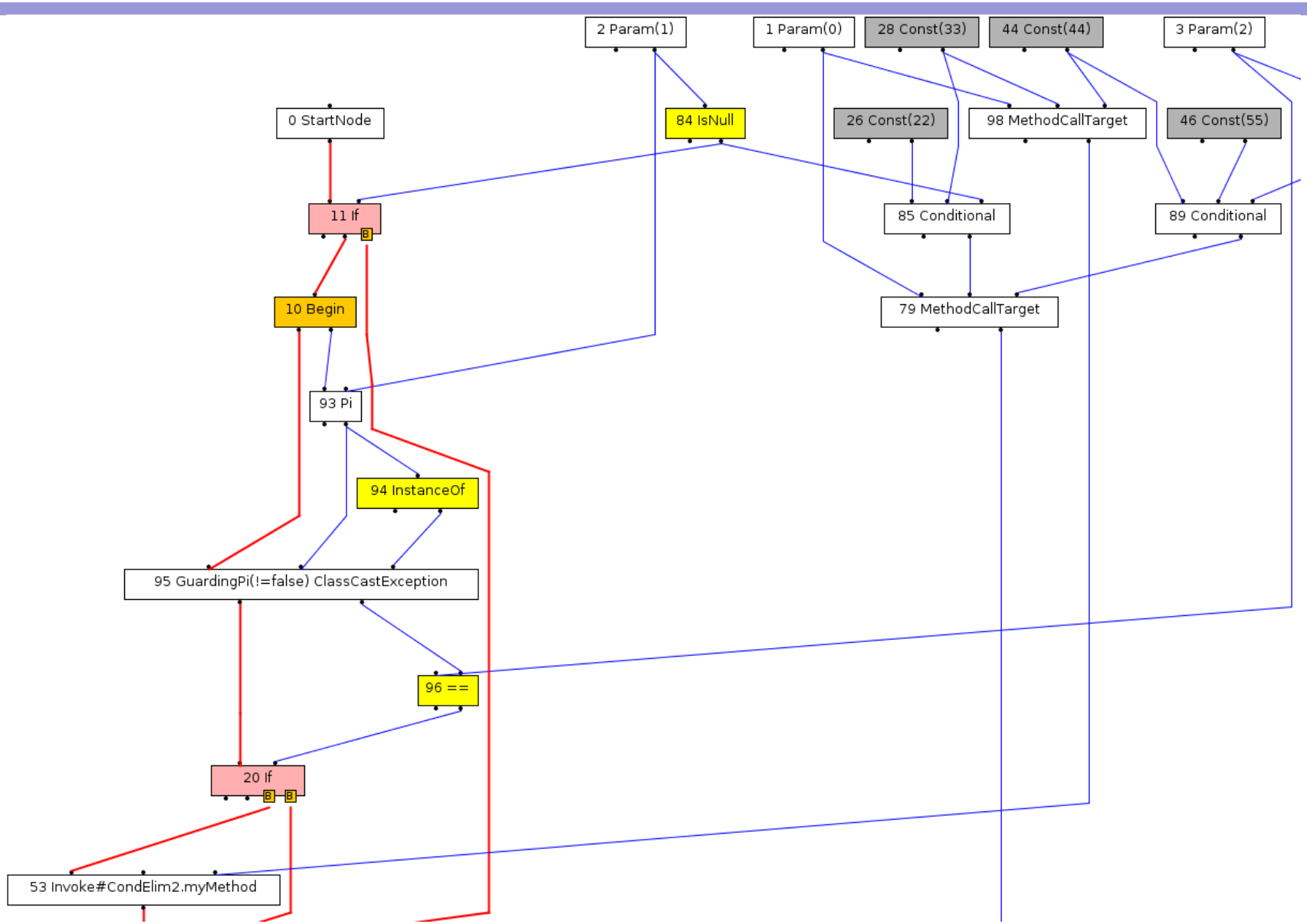
Miguel Garcia
http://lampwww.epfl.ch/~magarcia/

2014-07-07

Outline

Let's find redundancies in the program below:

```
public void t3Snippet(Object p, Object q) {
    if (p != null) {
        String s = (String) p;
        if (q == p) {
            myMethod(
               p == null            ? 22 : 33,
               q instanceof String ? 44 : 55
            );
        }
    }
    myMethod(
      p == null            ? 22 : 33,
      q instanceof String ? 44 : 55
    );
}
```


2 Param(1)
1 Param(0)
28 Const(33)
44 Const(44)
3 Param(2)
0 StartNode
84 IsNull
26 Const(22)
98 MethodCallTarget
46 Const(55)
11 If
85 Conditional
89 Conditional
10 Begin
79 MethodCallTarget
93 Pi
94 InstanceOf
95 GuardingPi(!=false) ClassCastException
96 ==
20 If
53 Invoke#CondElim2.myMethod

Goals of `FlowSensitiveReductionPhase`:

- Type Refinements:
  - PiNodes for reference values (most precise safe “downcasting”)
  - at receivers of virtual calls.
- Partial Evaluation:
  - reduction of side-effects free (multi-node) expressions,
  - whenever sub-expressions are known constant.

For now under a flag but works reliably. Check for yourself:

```
buildbot try -b try-dacapo_graal_sunfire ↩
  --properties="tryvmargs=-G:+FlowSensitiveReduction"

buildbot try -b try-scala-dacapo_graal_sunfire ↩
  --properties="tryvmargs=-G:+FlowSensitiveReduction"
```

`FlowSensitiveReductionPhase`

1. makes a single pass in dominator-based order over the graph
2. tracks flow-sensitive properties of values
   - via a state abstraction
   - for each reachable control-flow path
3. performs rewritings (type refinements, partial evaluation)

Details follow ...

"Track properties of values" means, update the state abstraction at the following fixed-nodes:

- two control-splits: `IfNode` and `TypeSwitchNode`
- check-casts
- guarding-pis
  (change the stamp of its input upon a condition being true)
- null-checks
- fixed-guards,

```
public FixedGuardNode(LogicNode condition,
                      DeoptimizationReason deoptReason,
                      DeoptimizationAction action)
```

At a program point, given its state abstraction, rewritings are performed:

- simplification of side-effects free expressions
  - inputs of fixed-nodes
  - devirtualization
- simplification of control-flow
  - input-condition to an `IfNode`
  - eliminating redundant check-casts, guarding-pis, null-checks, and fixed-guards, due to:
    - an equivalent guarding node already in scope: use it instead and remove the redundant one
    - "always fails": replace with `FixedGuardNode(false)`

Metrics obtained via `-G:+FlowSensitiveReduction -G:Meter=FlowSensitiveReduction`

Example: bootstrapping (after compiling 3418 methods):

- Cost

```
    |-> FSR-ImpossiblePathDetected=1250
    |-> FSR-NullnessRegistered=94540
    |-> FSR-ObjectEqualsRegistered=7765
    |-> FSR-TypeRegistered=78177
```

- Benefit

```
    |-> FSR-CheckCastRemoved=1
    |-> FSR-Downcasting=3045
    |-> FSR-EquationalReasoning=182
    |-> FSR-FixedGuardNodeRemoved=566
    |-> FSR-GuardingPiNodeRemoved=8212
    |-> FSR-InstanceOfRemoved=60
    |-> FSR-MethodResolved=19
    |-> FSR-NullCheckRemoved=1007
    |-> FSR-NullInserted=355
    |-> FSR-ObjectEqualsRemoved=0
    |-> FSR-UnconditionalDeoptInserted=1
```

Improvements over `ConditionalEliminationPhase`

- lower memory footprint: 3 maps instead of 6
- unreachable paths are detected and ignored, ie
  - no state updates on them
  - they don't get merged at join points, more precise merged states
- skips redundant cloning of state in SinglePassNodeIterator
  - `http://hg.openjdk.java.net/graal/graal/rev/9c9bb06a6b83`
  - `http://hg.openjdk.java.net/graal/graal/rev/84cf47e9c9f3`
- SinglePassNodeIterator: don't hold objects it won't access again
  - `early pruning of state map, visit a whole method`

  `http://hg.openjdk.java.net/graal/graal/rev/4d5b1e7a4d93`
- could be more compact with primitive-specialized collections
- tests, asserts, and source code documentation

Inliner improvements.

It all started with a few refactorings, 138 commits to be exact

Graph	Rev	Branch	Description	Author	Age
	16076	default	default tip **[inliner] de-duplicate parameters for callsites with duplicate arguments**	Miguel Garcia	21 hours
	16075	default	Merge	Miguel Garcia	26 hours
	16074	default	[inliner] documentation, more and better	Miguel Garcia	3 days
	16073	default	[inliner] extracted reusable query methods in InlineableGraph	Miguel Garcia	4 days
	16067	default	[inliner] documentation	Miguel Garcia	3 days
	16066	default	[inliner] documentation, more and better	Miguel Garcia	4 days
	16065	default	[inliner] extracted reusable query methods in InlineableGraph	Miguel Garcia	4 days
	16064	default	[inliner] documentation reorganization	Miguel Garcia	4 days
	16058	default	Merge	Miguel Garcia	5 days
	16053	default	[inliner] singleton pattern for DUMMY_CALLSITE_HOLDER	Miguel Garcia	5 days
	16052	default	[inliner] added a factory method in InlineInfo to make code uniform elsewhere	Miguel Garcia	5 days
	16051	default	[inliner] another mutator that finds its way to the class where it belongs	Miguel Garcia	5 days
	16050	default	[inliner] readability	Miguel Garcia	7 days
	16034	default	[inliner] the two personalities embodied by CallsiteHolder finally taken apart	Miguel Garcia	7 days
	16033	default	[inliner] assertion for result in a single place (producer) not at each consumer	Miguel Garcia	7 days
	16032	default	[inliner] moved helper method to CallsiteHolder	Miguel Garcia	8 days
	16000	default	Merge	Miguel Garcia	8 days
	15999	default	[inliner] no need to alias a final field	Miguel Garcia	8 days
	15998	default	[inliner] removed a method, lost nothing (but code is more readable afterwards)	Miguel Garcia	8 days
	15997	default	[inliner] trickle up, thus making more visible, graph copying	Miguel Garcia	8 days
	15996	default	[inliner] readability	Miguel Garcia	8 days
	15995	default	[inliner] both parts of what used to be a single method now invoked in sequence	Miguel Garcia	8 days
	15994	default	[inliner] break method up, to enable delaying specializeGraphToArguments()	Miguel Garcia	8 days
	15993	default	[inliner] documentation	Miguel Garcia	8 days
	15992	default	[inliner] return result versus parameter mutation, former deemed more readable	Miguel Garcia	8 days
	15991	default	[inliner] additional bits and pieces of documentation and assertions	Miguel Garcia	9 days
	15988	default	[inliner] lazy allocation of param-usages container; documentation	Miguel Garcia	9 days
	15987	default	[inlining] preparations to avoid cloning whenever possible	Miguel Garcia	11 days
	15986	default	[inlining] more uniform treatment of method cloning in InlineableGraph	Miguel Garcia	11 days
	15985	default	[inlining] refactoring for readability in InlineableGraph	Miguel Garcia	11 days
	15984	default	[inlining] made explicit an invariant of InliningData	Miguel Garcia	11 days
	15983	default	[inlining] documentation and assertions	Miguel Garcia	12 days
	15982	default	[probability-cache] documentation, assertions added; unreachable code removed	Miguel Garcia	13 days
	15860	default	[inlining] check maxMethodPerInlining after discarding methods below threshold	Miguel Garcia	2 weeks
	15859	default	[inlining] more precise type in createDispatchOnTypeBeforeInvoke()	Miguel Garcia	2 weeks
	15858	default	[inlining] isEmpty() favored over size() == 0	Miguel Garcia	2 weeks
	15857	default	[inlining] forgotten assertion, counterpart to the one in pushGraph()	Miguel Garcia	2 weeks
	15856	default	[inlining] operation that pushes invocation goes ahead and pushes graphs too	Miguel Garcia	2 weeks
	15855	default	[inlining] readability in CallsiteHolder constructor, part 2	Miguel Garcia	2 weeks
	15854	default	[inlining] readability in CallsiteHolder constructor, part 1	Miguel Garcia	2 weeks
	15815	default	[inlining] another renaming to avoid misleading type suggestion (2 of 2)	Miguel Garcia	3 weeks
	15814	default	[inlining] another renaming to avoid misleading type suggestion (1 of 2)	Miguel Garcia	3 weeks
	15813	default	[inlining] renaming to convey underlying types (2 of 2)	Miguel Garcia	3 weeks
	15812	default	[inlining] renaming to convey underlying types (1 of 2)	Miguel Garcia	3 weeks
	15811	default	[inline-info] step 3, InlineInfo leaves populateInlineInfo fully initialized	Miguel Garcia	3 weeks
	15810	default	[inline-info] step 2, simpler inter-procedural communication	Miguel Garcia	3 weeks
	15809	default	[inline-info] step 1 of de-aliasing MethodInvocation assumptions	Miguel Garcia	3 weeks
	15808	default	[inline-info] towards initializing InlineInfo in one place	Miguel Garcia	3 weeks
	15804	default	Merge	Miguel Garcia	3 weeks
	15803	default	[inlining-7] end of refactoring trail, helper methods now closer to users	Miguel Garcia	3 weeks
	15802	default	[inlining-7] InlineableGraph takes care of setup chores during construction	Miguel Garcia	3 weeks
	15801	default	[inlining-7] moved three utilities methods to where they belong	Miguel Garcia	3 weeks
	15800	default	[inlining-6] InlineableGraph now in package for inlineable elements	Miguel Garcia	3 weeks
	15799	default	[inlining-6] InlineableMacroNode now in package for inlineable elements	Miguel Garcia	3 weeks
	15798	default	[inlining-6] moved Inlineable to dedicated package for inlineable elements	Miguel Garcia	3 weeks

Graph	Rev	Branch	Description	Author	Age
	15797	default	[inlining-5] separate check code (fewer args, pure, concise) from logging code	Miguel Garcia	3 weeks
	15796	default	[inlining-5] "where does optimisticOpts come from?" answered	Miguel Garcia	3 weeks
	15795	default	[inlining-5] "where does replacements come from?" answered	Miguel Garcia	3 weeks
	15794	default	[inlining-5] checkTargetConditions() about to lose some of its formal params	Miguel Garcia	3 weeks
	15774	default	Merge	Miguel Garcia	3 weeks
	15773	default	[inlining-4] privatizing methods that can be made private	Miguel Garcia	3 weeks
	15772	default	[inlining-4] one less alias in getTypeCheckedInlineInfo()	Miguel Garcia	3 weeks
	15771	default	[inlining-4] one less alias in getAssumptionInlineInfo()	Miguel Garcia	3 weeks
	15770	default	[inlining-4] one less alias in getExactInlineInfo()	Miguel Garcia	3 weeks
	15769	default	[inlining-4] removed alias for InliningData.maxMethodPerInlining	Miguel Garcia	3 weeks
	15768	default	[inlining-4] getTypeCheckedInlineInfo() can get context.getReplacements() itself	Miguel Garcia	3 weeks
	15767	default	[inlining-4] getAssumptionInlineInfo() can get context.getReplacements() itself	Miguel Garcia	3 weeks
	15766	default	[inlining-4] no need to pass context.getReplacements() to getExactInlineInfo()	Miguel Garcia	3 weeks
	15765	default	[inlining-4] parameter aliasing context.getOptimisticOptimizations() goes away	Miguel Garcia	3 weeks
	15764	default	[inlining-4] parameter aliasing context.getReplacements() goes away	Miguel Garcia	3 weeks
	15763	default	[inlining-4] the method param that aliased maxMethodPerInlining goes away	Miguel Garcia	3 weeks
	15762	default	[inlining-4] getTypeInlineInfo() becomes instance method of InliningData	Miguel Garcia	3 weeks
	15761	default	[inlining-4] getTypeCheckedInlineInfo() becomes instance method of InliningData	Miguel Garcia	3 weeks
	15760	default	[inlining-4] getAssumptionInlineInfo() becomes instance method of InliningData	Miguel Garcia	3 weeks
	15759	default	[inlining-4] start of refactoring trail, by the end shorter parameter lists	Miguel Garcia	3 weeks
	15758	default	[inlining-3] readability of checkInvokeConditions() part 2 of 2	Miguel Garcia	3 weeks
	15757	default	[inlining-3] readability of checkInvokeConditions() part 1 of 2	Miguel Garcia	3 weeks
	15753	default	Merge	Miguel Garcia	3 weeks
	15752	default	[inlining-2] no guesswork at callsites about return value of logNotInlinedMethod	Miguel Garcia	3 weeks
	15751	default	[inlining-2] no guesswork about return value of logInlinedMethod (2/2)	Miguel Garcia	3 weeks
	15750	default	[inlining-2] no guesswork about return value of logInlinedMethod (1/2)	Miguel Garcia	3 weeks
	15749	default	[inlining-2] logInliningDecision, for side-effects not return value (2/2)	Miguel Garcia	3 weeks
	15748	default	[inlining-2] logInliningDecision, for side-effects not return value (1/2)	Miguel Garcia	3 weeks
	15747	default	[inlining-2] logNotInlinedMethod invoked only for side-effects not return value	Miguel Garcia	3 weeks
	15746	default	[inlining-2] make explicit the value returned by logNotInlinedInvoke()	Miguel Garcia	3 weeks
	15745	default	[inlining-2] renaming logNotInlinedMethodAndReturnNull -> logNotInlinedInvoke	Miguel Garcia	3 weeks
	15744	default	[inlining-2] make returned value explicit	Miguel Garcia	3 weeks
	15743	default	[inlining-2] renaming of an overloaded method	Miguel Garcia	3 weeks
	15740	default	[inlining-2] pulling side-effects (logging) out of method that evals a condition	Miguel Garcia	3 weeks
	15739	default	[inlining-2] one less logging method to worry about	Miguel Garcia	3 weeks
	15738	default	[inlining-2] replaced method body with call to code duplicate	Miguel Garcia	3 weeks
	15737	default	[inlining-2] reduced verbosity in checkTargetConditions()	Miguel Garcia	3 weeks
	15736	default	[inlining-2] typos in source comment	Miguel Garcia	3 weeks
	15735	default	[inlining-2] consumer becomes initializer of the probabilities map	Miguel Garcia	3 weeks
	15734	default	[inlining-2] fixing input as instance final rather than passing it over and over	Miguel Garcia	3 weeks
	15721	default	[inlining] reverting refactoring trail until spoiling commit(s) are discovered	Miguel Garcia	3 weeks
	15720	default	[inlining] behavior becomes less argument-dependent, arguments become redundant	Miguel Garcia	3 weeks
	15719	default	[inlining] no need for guessing a return value that doesn't matter	Miguel Garcia	3 weeks
	15718	default	[inlining] untangling concerns, micro-step by micro-step	Miguel Garcia	3 weeks
	15717	default	[inlining] readability by means of import static	Miguel Garcia	3 weeks
	15716	default	[inlining] redux, no need for the suspense about return value	Miguel Garcia	3 weeks
	15715	default	[inlining] no need for the suspense about return value	Miguel Garcia	3 weeks
	15714	default	[inlining] another case of logNotInlined vs logNotInlinedMethod	Miguel Garcia	3 weeks
	15713	default	[inlining] shorter and equally informative, logNotInlined vs logNotInlinedMethod	Miguel Garcia	3 weeks
	15712	default	[inlining] "return null" favored again over "return ...AndReturnNull"	Miguel Garcia	3 weeks
	15711	default	[inlining] tradeoff: "return null" still shorter than "return ...AndReturnNull"	Miguel Garcia	3 weeks
	15710	default	[inlining] pulling side-effects (logging) out of method that evals a condition	Miguel Garcia	3 weeks
	15709	default	[inlining] one less logging method to worry about	Miguel Garcia	3 weeks
	15708	default	[inlining] replaced method body with call to code duplicate	Miguel Garcia	3 weeks
	15707	default	[inlining] reduced verbosity in checkTargetConditions()	Miguel Garcia	3 weeks

Cost/benefit of refactoring (a matter of when, not if)

- most of them “just” for readability
- more maintainable code, fewer errors in the long run
- but also for performance

Example (1 of 4): De-aliasing

- method-param aliases final-field

```
[inliner] no need to alias a final field

http://hg.openjdk.java.net/graal/graal/rev/096848853662
```

- method-param pointing to what’s navigable via final-field

```
[inlining-5] “where does optimisticOpts come from?”  answered

http://hg.openjdk.java.net/graal/graal/rev/ab2858ab79e9
```

- Such aliases contributed to the Inlining phase needing too many parameters

Example (2 of 4): moving mutators closer to the locations they mutate

- `[inliner] another mutator that finds its way to the class where it belongs`

  `http://hg.openjdk.java.net/graal/graal/rev/1461d7627707`

Example (3 of 4): Pull side-effects out gives reusable query-methods

- `pulling side-effects (logging) out of method that evals a condition`

  `http://hg.openjdk.java.net/graal/graal/rev/acfcb5ace52f`

Example (4 of 4): Performance (little things add up)

- `[inliner] lazy allocation of param-usages container; documentation`

  `http://hg.openjdk.java.net/graal/graal/rev/5aaef6a8985d`

- `[inliner] de-duplicate parameters for callsites with duplicate arguments`

  `http://hg.openjdk.java.net/graal/graal/rev/8d0202b354fb`

As refactoring progressed, synergy with:

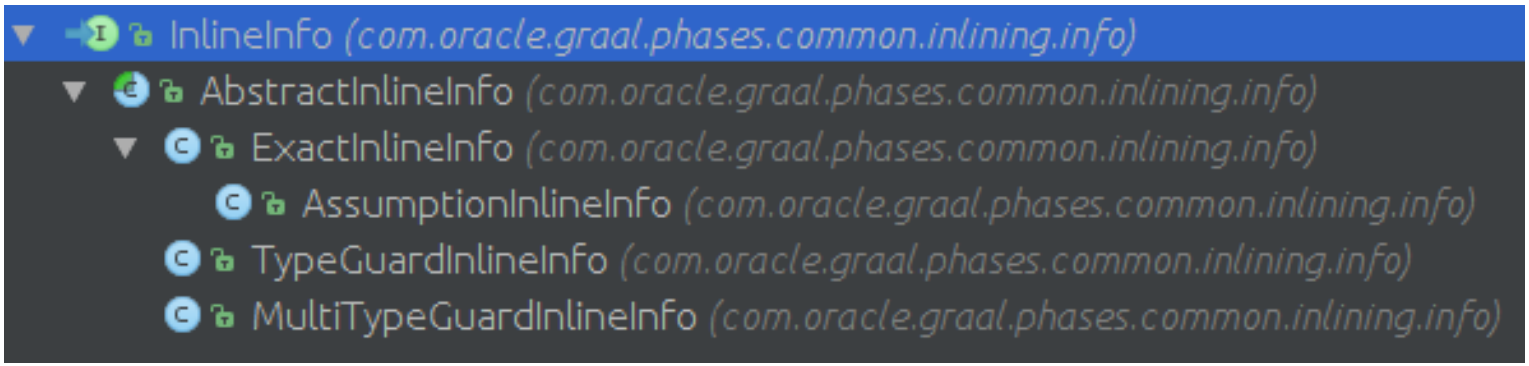
- Documentation
  - `[inlining] documentation`

    `http://hg.openjdk.java.net/graal/graal/rev/e90ec3e5e45b`
- Assertions
  - `[probability-cache] assertions added; unreachable code removed`

    `http://hg.openjdk.java.net/graal/graal/rev/aa28d876651a`

How the inliner works. It grows and shrinks a stack as follows:

1. Inlining candidates are explored depth-first,
   - sifting down caller information about arguments (eg, constants)
   - thus specializing the feasible target(s)
2. At some point exploration stops,
   - `InliningPolicy.continueInlining(StructuredGraph)`
3. Inlining: at a callsite in the stack-top graph



After some rounds of the above,

- the stack of inlining candidates eventually shrinks back
- to the root method on which inlining was launched

Customization point (1 of 2)

Pick a “promising” callsite first (as next candidate for exploration during depth-first search):

```
public static class ArgumentStats
implements Comparable<ArgumentStats> {
    // The immutable positions of freshly instantiated arguments
    public final BitSet freshArgsBitSet;
    public final boolean hasFixedReceiver;
    public final int sizeConstantArgs;
    public final int sizeFreshArgs;
```

A freshly-instantiated argument is either:

- an `AbstractNewObjectNode`, `AllocatedObjectNode`, or `VirtualObjectNode`
- a `ParameterNode` whose corresponding argument is freshly-instantiated.

Why “promising”?

- Once inlined, freshly-instantiated args (usually) don’t escape
- field accesses on freshly-instantiated args avoid indirection

Realized in terms of:

- `[inliner] propagating fresh-instantiation info through call-hierarchy`

  `http://hg.openjdk.java.net/graal/graal/rev/f98b033b6050`
- a priority-queue for `CallsiteHolderExplorable.remainingInvokes`, comparing `ArgumentStats`

However, the above alone didn’t improve performance.

- ToDo: priority-queue should also consider frequency and relevance, as in `InliningPolicy`

Customization point (2 of 2)

- Each feasible target method (receiver type-profiling) is explored before inlining (ie, its graph is inspected)
- Implementation-wise, an `InlineableGraph` is built for that target
- ToDo: Detect usages of freshly-instantiated args to influence inlining decisions.

Motivation:

```
for (i <- 1 to 10) { i => closure-body }
```

desugared to

```
new Range(1, 10, 1).foreach(new MyClosure(...captured-values...))
```

Similarly, Command pattern in Java:

```
Collections.sort(clazzes, new Comparator<ResolvedJavaType>() {
    @Override
    public int compare(ResolvedJavaType o1, ResolvedJavaType o2) {
        ...
```

Back to the for-loop example. The `foreach` receiver is an instance of:

```
class Range(val start: Int, val end: Int, val step: Int)
extends scala.collection.AbstractSeq[Int] with ...
{
  ...
  /*- this method may well get invoked only once
      per activation of its containing method,
      thus not necessarily "hot" */
  @inline final override def foreach[@specialized(Unit) U](f: Int => U) {
    validateMaxLength()
    val isCommonCase = (start != Int.MinValue || end != Int.MinValue)
    var i = start
    var count = 0
    val terminal = terminalElement
    val step = this.step
    while(
      if(isCommonCase) { i != terminal }
      else { count < numRangeElements }
    ) {
      f(i) /*- <-------- closure invocation inside a loop */
      count += 1
      i += step
    }
  }
```

Side note: allowing loops in expressions sometimes trigger dreaded:
COMPILE SKIPPED: OSR starts with non-empty stack (not retryable)

Callee-graph specialization: never too soon

The graph that gets pushed for a feasible target is already specialized to the arguments at the callsite (must be — why in a moment):

- constants replace the corresponding `ParameterNode`
- duplicate arguments result in a single representative `ParameterNode` being used for all duplicates
- the more precise stamp of an argument is carried over to its `ParameterNode` (in particular for freshly-instantiated args)

Any missed opportunity to specialize the callee-graph results in missed follow-on reductions at all nested invocation levels.

Terminology:

freshly-instantiated arg (caller perspective)
and
fixed-param (callee perspective)
are two sides of the same coin

Missed opportunities:

1. final-fields of fixed-params denote a caller-available SSA value:
   sometimes a constant
   sometimes a value with more precise stamp
2. in the other direction, caller-specialization suggested by callee-graph:
   - more specific return type
   - constant return value

   That would be good to know, because:
   - allows callsite specialization without inlining (just downcast)
   - follow-on reductions throughout the caller

In the example

```
new Range(1, 10, 1).foreach{ i => ... }
```

which final-fields are amenable to the optimization above?

Note: the “loop body” may include captured-usages as well as further higher-order expressions; compounding the “missed opportunities” bill.

A word of caution about “callsite-specific return-types”, quoting from

http://mail.openjdk.java.net/pipermail/graal-dev/2014-January/001538.html

The GraphBuilder is not built for precise analyses of bytecode methods - consider this example:

```
Object foo (int a) {
        try { return Integer.valueOf(1000 / a); }
        catch (ArithmeticException e) { return new Error(); }
}
```

The GraphBuilder never generates code to handle the division by zero, so the exception handler will never be compiled, even with all optimistic optimizations disabled. Therefore, information about the return type would need to be checked before being used.

Counterpoint to
propagating “final-fields of fixed-parms” from caller to callee:

1. No problem,
   - after inlining kicks in
   - read-elimination will propagate (constants, etc) anyway.

   Except that, inlining less likely to kick in
   for non-specialized callee-graph (chicken and egg).
2. Not always safe to propagate instance-final fields.

Summary of forces at play (next slide)

Quoting from http://www.azulsystems.com/blog/cliff/2011-10-27-final-fields-part-2

These other frameworks are doing a Read (and if it is null), a Write [via reflection], then futher Reads. ... when its JIT'd [null is] the value used for some of the later Reads.

...

To summarize: JRuby makes lots of final fields that really ARE final, and they span not-inlined calls (so require the final moniker to be CSE'd), AND such things are heavily chained together so there's lots of follow-on CSE to be had.

...

If I turn off final field optimizations to save the Generic Popular Frameworks, I burn JRuby and probably other non-Java languages that emit non-traditional (but legal) bytecodes. If I don't turn them off, these frameworks take weird NULL exceptions under load (as the JIT kicks in).

Future Work

1. lazy graph-copying of inlining candidates
   (ie, copy-on-write, or equivalently: no mutation – no copying)
2. take into account usages of freshly-instantiated args

Next in line, please!: exploiting the indirect benefits of inlining by accurately predicting further inlining

Full Text: PDF

Authors: Andreas Sewe Technische Universität Darmstadt, Darmstadt, Germany
Jannik Jochem Technische Universität Darmstadt, Darmstadt, Germany
Mira Mezini Technische Universität Darmstadt, Darmstadt, Germany

2011 Article

Published in:

· Proceeding
SPLASH '11 Workshops Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, & VMIL'11
Pages 317-328
ACM New York, NY, USA ©2011
table of contents ISBN: 978-1-4503-1183-0 doi>10.1145/2095050.2095102

Bibliometrics

· Downloads (6 Weeks): 6
· Downloads (12 Months): 25
· Downloads (cumulative): 55
· Citation Count: 1