

The backend that makes `scalac` faster for real

Miguel Garcia

<http://lamp.epfl.ch/~magarcia>

LAMP, EPFL

2013-04-11

Outline

Background

- Why a new optimizer

- Workflow of the new optimizer

Under the hood

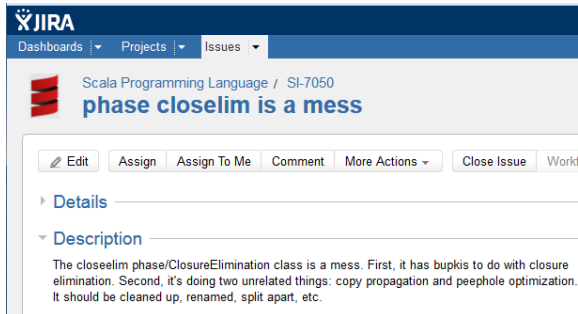
- Bytecode emitter (GenBCode)

- Intra-method optimizations

- More compact code

Lessons learnt about speeding up the compiler

- ▶ you may have heard `scalac` could be faster
- ▶ maintainability, upgradability of the current optimizer. Quote:

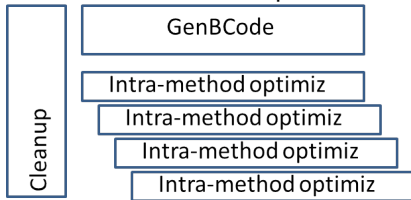


The screenshot shows a JIRA issue page. At the top, there's a navigation bar with 'Dashboards', 'Projects', and 'Issues'. Below that, the issue title is 'phase closelim is a mess' under the project 'Scala Programming Language / SI-7050'. There are several action buttons: 'Edit', 'Assign', 'Assign To Me', 'Comment', 'More Actions', 'Close Issue', and 'Work'. The 'Description' section is expanded, showing the text: 'The closelim phase/ClosureElimination class is a mess. First, it has bukis to do with closure elimination. Second, it's doing two unrelated things: copy propagation and peephole optimization. It should be cleaned up, renamed, split apart, etc.'

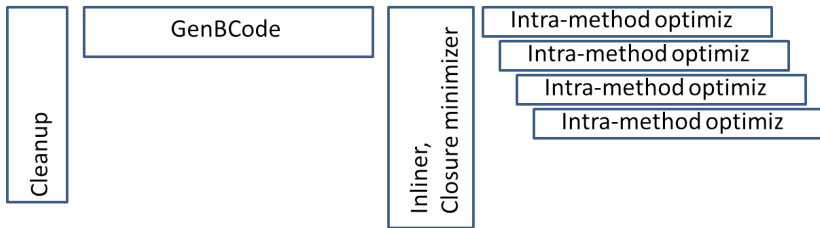
The new backend, <http://magarciaepfl.github.io/scala/>

- ▶ makes the compiler 15% faster (uptime, also lowering CPU time)
- ▶ emits 10% more compact code (and that's without `-neo:01`)
- ▶ documented
- ▶ future-proof (ASM-based, e.g. in view of Java 8)

Level 1: Intra-method optimizations



Levels 2 and 3: Intra-program and Cross-libraries, resp.



Simplicity of each component contributes to overall simplicity

AST ClassDef node → ASM ClassNode easier than expected:

- ▶ CFG not necessary
- ▶ just visit pre-order an expression node. Gist:

```
def genNormalMethodCall() {  
  
  if (invokeStyle.hasInstance) { genLoadQualifier(fun) }  
  
  genLoadArguments(args, paramTKs(app))  
  
  // ASM visitMethodInsn(Opcodes.INVOKEVIRTUAL, owner, name, desc)  
}
```

Additionally, *not seen before* features like:

- ▶ more thorough outer-pointer elimination, [https:](https://github.com/magarciaEPFL/scala/commit/0a426b640411ee85983a6deb8a5612ebaa6d5ff3)

[//github.com/magarciaEPFL/scala/commit/0a426b640411ee85983a6deb8a5612ebaa6d5ff3](https://github.com/magarciaEPFL/scala/commit/0a426b640411ee85983a6deb8a5612ebaa6d5ff3)

- ▶ method handles, to cut down on anonymous-closure classes
- ▶ distinction between intra-program and cross-library inlining

▶ Control-flow simplifications

- ▶ collapse a multi-jump chain to target its final destination via a single jump
- ▶ remove unreachable code
- ▶ nullness propagation

▶ Propagation of known values





- ▶ copy propagation
- ▶ dead-store elimination
- ▶ Preserve side-effects, but remove those (producer, consumer) pairs where the consumer is a DROP and the producer has its value consumed only by the DROP in question.
- ▶ constant folding
- ▶ eliding box/unbox pairs
- ▶ eliding redundant local vars

Contrasting how the old and new optimizer tackle the same problem

<https://github.com/scala/scala/pull/2214>

Just a few examples:

- ▶ method `driver()` in `test/files/run/t7181.scala`
 - ▶ 881 instructions, after `-neo:o2 -closurify:delegating`
 - ▶ 1004 instructions, with GenASM and `-optimise`

 <code>UnoptCompByBCode.jar</code>	11,875 KB
 <code>UnoptCompByICode.jar</code>	13,353 KB
 <code>UnoptLibByBCode.jar</code>	6,473 KB
 <code>UnoptLibByICode.jar</code>	6,746 KB

“Late Closure Classes” was the single largest contributor to the 15% speedup, because it results in less work for lambda lift, specialize, and erasure, among others.

```

/*
 * Transform a function node (x_1, ..., x_n) => body
 * of type FunctionN[T_1, .., T_N, R] to a Block
 *
 * {
 *   def hoisted(x_1: T_1, ..., x_N: T_n): R = body
 *
 *   hoisted(zeroes-for-params-above).asInstanceOf[AbstractFunctionN[
 * }
 *
 * The bytecode emitter will either:
 * (a) emit an anonymous closure class and its instantiation; or
 * (b) emit a method handle given as
 *     constructor-argument to a closure instantiation.
 */
def closureConversionModern(fun: Function): Tree = {

```