On the Performance Characteristics of Scala Programs on the Java Virtual Machine

Abstract

In recent years, the Java Virtual Machine has become an attractive target for a multitude of programming languages, one of which is Scala. But while the Scala compiler emits plain Java bytecode, the performance characteristics of Scala programs are not necessarily similar to those of Java programs. We therefore propose to complement a popular Java benchmark suite with several Scala programs and to subsequently evaluate their performance using VM-independent metrics.

1 Towards a Scala Benchmark Suite

Previous investigations into the performance of Scala programs have been mostly restricted to micro-benchmarking. (The language’s implementers themselves perform a number of so-called shoot-outs, each testing a particular language feature) While undesirably useful to the implementers of the Scala compiler, such micro-benchmarks are less useful to implementers of a Java VM, who have to deliver good performance across a wide range of real-world programs—only some of which are written in Scala. Thus, a full-fledged benchmark suite consisting of both Scala and Java programs is needed.

The following programs (along with potential input data) have been selected for inclusion in our Scala benchmark suite, developed as an extension to the popular DaCapo suite [2].

- The Lift web framework, which uses Scala’s actor library to
- The Stanford Topic Modeling Toolbox, a natural language
- ScalaTest, a testing framework supporting various
- Scalaz, an extension library.
- The Kiama library for language processing (compiling

The following programs were also included as Java versions. They are not included in our VM-independent measurements, and their results are not shown here.

2 Towards VM-Independent Benchmark Comparisons

In order to make general claims about the similarities of Scala and Java with respect to performance, VM-independent metrics are needed. Moreover, these metrics need to be relevant in the sense that they correlate with either optimisation opportunities themselves or with the cost of exploiting said opportunities. Two such metrics are related in the following figure.

All of the above measurements were conducted using JP [1], a tool for VM-independent, complete calling-context profiling. (Due to technical limitations, momentarily only a subset of benchmarks is covered.)

3 Future Directions

As the semantic gap between Scala source code and Java bytecode is wider than the gap between Java source and bytecode, the trade-offs involved in the optimising compiler vs. optimising VM decision needs to be investigated anew. Unlike the Java compiler, the Scala compiler already performs several optimisations on its own: method inlining, escape analysis (for closure elimination), and tail call optimisation. It is an open question, however, whether the semantic gap is wide enough to warrant such re-implementations of optimisations within the compiler or whether the VM remains the proper place for optimisations.

Also, Scala targets a second platform besides the JVM, namely the Common Language Runtime (CLR). This choice, JVM vs. Common Language Runtime, offers the unique opportunity to put all our findings to the test: Are they specific to a single platform or is a generalisation possible?

References


Acknowledgments

Contact Information

Andreas Sewe
sewe@st.informatik.tu-darmstadt.de
Technische Universität Darmstadt
Hochschulstr. 10
64289 Darmstadt, Germany