

Dynamic Compilation of DSLs

Vojin Jovanovic

EPFL, Switzerland

vojin.jovanovic@epfl.ch

Domain-specific language (DSL) compilers use *domain knowledge* to perform *domain-specific optimizations* that can yield several orders of magnitude speedups [4]. These optimizations, however, often require knowledge of the run-time values. For example, in matrix-chain multiplication, knowing matrix sizes allows choosing the optimal multiplication order [2, Ch. 15.2] and in relational algebra knowing relation sizes is necessary for choosing the right join order [5]. As a motivating example we will use multiplication of three matrices in a DSL for linear algebra embedded in Scala:

```
val (m1, m2, m3) = ... // matrices of unknown size
m1 * m2 * m3
```

In this program, without knowing the matrix sizes, the DSL compiler can not determine the optimal order of multiplications. There are two possible orders $(m1*m2)*m3$ with an estimated cost $c1$ and $m1*(m2*m3)$ with an estimated cost $c2$ where:

```
c1 = m1.rows*m1.columns*m2.columns+m1.rows*m2.columns*m3.rows
c2 = m2.rows*m2.columns*m3.columns+m1.rows*m2.rows*m3.columns
```

Ideally we would change the multiplication order at runtime only when the condition $c1 > c2$ changes. The only solution for producing an optimal program is to use *dynamic compilation* [1].

Existing dynamic compilation systems, however, are not a right fit for this task. They use run-time information for transformations that are not domain specific, e.g., specialization [3], loop unrolling [6], etc. In these systems:

- Value profiling tracks the *stability* of values. In our example we need to track stability over computation over several (possibly unstable) values that are used for computing $c1$ and $c2$.
- Guards for recompilation are implemented as equality checks between the current value and the previous stable value of the variable. In our example we need comparisons with previous versions of the computation over run-time values ($c1 > c2$).
- Code caches are based on checking equality of stable values. In our example, we need equality with computed values that change the outcome of DSL compilation.

Only exception to existing dynamic compilation systems is Truffle [7] that allows creation of custom profiling and recompilation guards. However, with Truffle, language designers do not have the full view of the program, and thus,

can not perform global optimizations (e.g., matrix chain multiplication optimization).

We propose a dynamic compilation system aimed for domain specific languages where:

- DSL authors declaratively, at the definition site, state the values that are of interest for dynamic compilation (e.g., array and matrix sizes, vector and matrix sparsity, etc.). These values can regularly be used for making compilation decisions throughout the DSL compilation pipeline.
- The instrumented DSL compiler transparently reifies all computations on the run-time values that will affect compilation decisions. In our example, the compiler reifies and stores all computations on run-time values in the unmodified dynamic programming algorithm [2] for determining the optimal multiplication order (i.e., $c1 > c2$).
- Recompilation guards are introduced automatically based on the stored DSL compilation process (i.e., recompilation guard is $c1 > c2$ in the example).
- Automatically manages code caches that are addressed with outcomes of the DSL compilation decisions instead with stable individual values. In the example the code cache would have two entries addressed with a single boolean value computed with $c1 > c2$.

The goal of this talk is to foster discussion on the new approach to dynamic compilation with focus on different policies for automatic introduction of re-compilation guards: *i)* heuristic, *ii)* *DSL author specified*, and *based on domain knowledge*.

References

1. Joel Auslander, Matthai Philipose, Craig Chambers, Susan J Eggers, and Brian N Bershad. Fast, effective dynamic compilation. In *International Conference on Programming Language Design and Implementation (PLDI)*, 1996.
2. Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, et al. *Introduction to algorithms*, volume 2. MIT press Cambridge, 2001.
3. Brian Grant, Markus Mock, Matthai Philipose, Craig Chambers, and Susan J Eggers. DyC: an expressive annotation-directed dynamic compiler for C. *Theoretical Computer Science*, 248(1), 2000.
4. Tiark Rompf, Arvind K. Sujeeth, Nada Amin, Kevin J. Brown, Vojin Jovanović, HyoukJoong Lee, Manohar Jonnalagedda, Kunle Olukotun, and Martin Odersky. Optimizing data structures in high-level programs: New directions for extensible compilers based on staging. In *Symposium on Principles of Programming Languages (POPL)*, 2013.
5. P Griffiths Selinger, Morton M Astrahan, Donald D Chamberlin, Raymond A Lorie, and Thomas G Price. Access path selection in a relational database management system. In *International conference on Management of data (SIGMOD)*, pages 23–34, 1979.
6. Toshio Sukanuma, Toshiaki Yasue, Motohiro Kawahito, Hideaki Komatsu, and Toshio Nakatani. A dynamic optimization framework for a java just-in-time compiler. In *International Conference on Programming Language Design and Implementation (PLDI)*, volume 36, pages 180–195. ACM, 2001.

7. Thomas Würthinger, Christian Wimmer, Andreas Wöß, Lukas Stadler, Gilles Duboscq, Christian Humer, Gregor Richards, Doug Simon, and Mario Wolczko. One VM to rule them all. In *Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software (Onward!)*, 2013.