Truffle: A Self-Optimizing Runtime System

Christian Wimmer Thomas Würthinger

Oracle Labs

christian.wimmer@oracle.com thomas.wuerthinger@oracle.com

Abstract

We present Truffle, a novel framework for implementing managed languages in Java™. The language implementer writes an AST interpreter, which is integrated in our framework that allows tree rewriting during AST interpretation. Tree rewrites incorporate type feedback and other profiling information into the tree, thus specializing the tree and augmenting it with run-time information. When the tree reaches a stable state, partial evaluation compiles the tree into optimized machine code. The partial evaluation is done by Graal, the just-in-time compiler of our Java VM (a variation of the Java HotSpot VM). To show that Truffle supports a variety of programming language paradigms, we present prototype implementations of JavaScript (a dynamically typed programming language) and J (an array programming language).

Categories and Subject Descriptors D.3.4 [*Programming Languages*]: Processors—Run-time environments, Optimization

General Terms Algorithms, Languages, Performance

Keywords Java, JavaScript, J, Truffle, Graal, dynamic languages, virtual machine, language implementation

1. Introduction

An abstract syntax tree (AST) interpreter is a simple and natural way to implement a programming language. However, it is usually also considered the slowest approach because of the high overhead of virtual method dispatch. Language implementers therefore define bytecodes to speed up interpretation, followed by a just-in-time compiler that is needed to reach excellent peak performance. In addition, a high-performance garbage collector is necessary for automatic memory management, together with a runtime system to form a complete virtual machine (VM) The algorithms for all these components are well known. However, VM code is rarely reused when implementing a VM for a new language. This makes the process of developing new high-performance languages expensive and tedious.

Truffle is a novel approach to implement AST interpreters in which the syntax tree is modified during interpretation to incorporate type feedback [5]. This tree rewriting is a general and powerful mechanism to optimize many constructs common in dynamic programming languages. Our system is implemented in Java and uses the static typing and primitive data types of Java elegantly to avoid a boxed representation of primitive values in dynamic programming languages.

The just-in-time (JIT) compiler of our Java VM, named Graal [3], is extensible and accessible from the AST interpreter. The Truffle compilation system uses Graal on top of the Java HotSpot VM to create optimized machine code snippets for parts of a Truffle AST. The main idea is to exploit that the AST changes rarely after it has reached a stable state. We use partial evaluation and assume that the current AST node structure remains constant. We inline the `execute` methods of all AST nodes into one big compilation unit. This compilation unit is then optimized by Graal. If there is a control path that would change the Truffle AST, we remove it from the compiled code and instead replace it with a runtime call that triggers deoptimization [2], i.e., the optimized machine code is discarded and execution continues in the AST interpreter. This way, we are able to create optimized machine code for a Truffle sub-tree that only contains the fast path for every node. This results in an executable version of the sub-tree that is valid as long as no AST rewriting needs to be performed. Such rewriting should be relatively rare as an AST will only be scheduled for optimization when profiling indicates that it is stable.

The source code of our system is available as open source from [3].

2. Related Work

The system most closely related to Truffle is PyPy [4]. Truffle shares with PyPy the main mission: automatically deriving an efficient implementation of a language by using an interpreter of that language written in a statically typed language. The difference between PyPy and Truffle is that PyPy uses a trace-based JIT compiler [1], while Truffle uses a traditional method-based JIT compiler. Because of the dy-

Copyright is held by the author/owner(s).

SPLASH'12, October 19–26, 2012, Tucson, Arizona, USA.
ACM 978-1-4503-1563-0/12/10.

namic AST replacements in Truffle, we can leverage the best of a trace compiler, i.e., only compiling the specialized fast paths, while at the same time avoiding many of the problems of trace compilers, e.g., handling or recursive method calls, complications from trace tree merging to avoid code explosion, or trace recording overhead.

3. Demonstration Outline

- Motivation for a modular language framework and re-use of VM components.
- System architecture: Truffle runs on the Graal VM, a modified version of the Java HotSpot VM with the Graal JIT compiler. We present the basic architecture of Graal.
- Overview of Truffle: core classes that form the AST and the framework for AST rewriting.
- AST interpreter cookbook: implementation of an AST interpreter for a simple language. We will implement a language and integrate it with the Truffle framework to get a high-performance implementation of this language.
- Overview of the existing language implementations that are under development: JavaScript (a dynamically typed programming language) and J (an array programming language).

4. Presenters

Christian Wimmer is a researcher at Oracle Labs, working on the Maxine VM, the Graal compiler, the Truffle dynamic language infrastructure, as well as on other projects that involve dynamic compilation and optimizations. His research interests span from compilers, virtual machines, and secure systems to component-based software architectures. He received a Dr. techn. degree in Computer Science (advisor: Prof. Hanspeter Mössenböck) and a Dipl.-Ing. degree in Computer Science, both from the Johannes Kepler University Linz, Austria. Before the time at Oracle, he was a postdoctoral researcher at the Department of Computer Science of the University of California, Irvine, working with Prof. Michael Franz.

Thomas Würthinger is a researcher at Oracle Labs in Austria. He works on the Graal compiler, the Truffle dynamic language infrastructure, and other projects in the area of virtual machines. His research interests include compilers, virtual machines, and graph visualization. He received a Dr. techn. degree in Computer Science (advisor: Prof. Hanspeter Mössenböck) and a Dipl.-Ing. degree in Computer Science, both from the Johannes Kepler University Linz, Austria.

Acknowledgments

Truffle and Graal would not be possible without the efforts of our academic collaborators, especially the Institute for System Software at the Johannes Kepler University Linz. We would also like to thank all members of the Virtual Machine Research Group at Oracle Labs for their support and contributions.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

References

[1] C. F. Bolz, A. Cuni, M. Fijałkowski, and A. Rigo. Tracing the meta-level: PyPy's tracing JIT compiler. In *Proceedings of the Workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages and Programming Systems*, pages 18–25. ACM Press, 2009. doi: 10.1145/1565824.1565827.

[2] U. Hölzle, C. Chambers, and D. Ungar. Debugging optimized code with dynamic deoptimization. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 32–43. ACM Press, 1992. doi: 10.1145/143095.143114.

[3] Oracle. OpenJDK: Graal project, 2012. URL http://openjdk.java.net/projects/graal/.

[4] A. Rigo and S. Pedroni. PyPy's approach to virtual machine construction. In *Companion to the ACM SIGPLAN Conference on Object Oriented Programming Systems, Languages, and Applications*, pages 944–953. ACM Press, 2006. doi: 10.1145/1176617.1176753.

[5] T. Würthinger, A. Wöss, L. Stadler, G. Duboscq, D. Simon, and C. Wimmer. Self-optimizing AST interpreters. In *Proceedings of the Dynamic Languages Symposium*. ACM Press, 2012.