Faster FastR through Partial Evaluation and Compilation

Michael Haupt[1,⋆], **Christian Humer**[2], **Mick Jordan**[1], **Prahlad Joshi**[3], **Jan Vitek**[3], **Adam Welc**[1], **Christian Wirth**[1], **Andreas Wöß**[2], **Mario Wolczko**[1], **Thomas Würthinger**[1]

1. Oracle Labs
2. Johannes Kepler University, Linz, Austria
3. Purdue University, West Lafayette, IN, USA
⋆Contact author: michael.haupt@oracle.com

Keywords: R Language Runtime, Java, R Performance, AST Interpretation, Partial Evaluation

FastR, first introduced at *useR! 2013* [3], is an implementation of the *R* programming language in *Java* [2]. It uses the concept of self-specialising abstract syntax tree (AST) interpretation [5]. In such interpreters, AST nodes replace themselves with nodes that are specialised for handling the types and data actually occurring during execution. This saves considerable time in the implementation of dynamically typed programming languages.

The implementation introduced in 2013 was a pure interpreter. We introduce the next version of *FastR*. The current implementation is based on Truffle [4]. Truffle is a framework for the implementation of specialising AST interpreters. Truffle-based language implementations transparently employ partial evaluation of specialised ASTs, and dynamic compilation, to obtain performance competitive with that of dedicated dynamic compilers.

The performance of some development versions of *FastR* running the b25 benchmarks and an *R* version of a subset of the Computer Language Benchmarks Game ("shootout") is, on average, more than an order of magnitude faster than the GNU R byte code interpreter, and significantly faster than the purely interpreted version of FastR. *FastR* is available as an open source project [1] under the terms and conditions of the GNU General Public License 2.

We will describe the status of the implementation and outline our plans for the future. An important long-term goal of the *FastR* project is to dispense with the need for implementing performance-critical parts of *R* applications in lower-level languages.

References

[1] BitBucket (2014). FastR project. `http://bitbucket.org/allr/fastr`.

[2] Kalibera, T., P. Maj, F. Morandat, and J. Vitek (2014). A Fast Abstract Syntax Tree Interpreter for R. In *Proceedings of the 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '14, New York, NY, USA, pp. 89–102. ACM.

[3] Kalibera, T., P. Maj, and J. Vitek (2013). R in Java: Why and How? In *The R User Conference, useR! 2013, Book of Contributed Abstracts*, pp. 111. `http://www.edii.uclm.es/~useR-2013/docs/useR2013_abstract_booklet.pdf`.

[4] Würthinger, T., C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko (2013). One VM to rule them all. In *Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software*, pp. 187–204. ACM.

[5] Würthinger, T., A. Wöß, L. Stadler, G. Duboscq, D. Simon, and C. Wimmer (2012). Self-optimizing AST interpreters. In *Proceedings of the 8th Symposium on Dynamic Languages*, DLS '12, New York, NY, USA, pp. 73–82. ACM.