Newspeak and Truffle: A Platform for Grace?

Stefan Marr
School of Computing
University of Kent
United Kingdom
s.marr@kent.ac.uk

Richard Roberts
School of Engineering and Computer Science
Victoria University of Wellington
rykardo.r@gmail.com

James Noble
School of Engineering and Computer Science
Victoria University of Wellington
kjx@ecs.vuw.ac.nz

Abstract

The Newspeak language had a strong influence on Grace's language design: Is a Newspeak interpreter a good starting point for a Grace implementation?

Truffle is a framework for interpreter implementation and leverages the Graal just-in-time (JIT) compiler to offer state-of-the-art performance to dynamic languages. But does it fulfill the promise, and at which cost?

This talk is a review of about two years of work on a Grace implementation based on SOMns called Moth. What did we achieve? How compliant is Moth with the specification? How much of SOMns can we reuse to support Grace? Does Truffle live up to the promise?

1 Why another Grace implementation?

Perhaps the first question to answer when deciding to implement a language is the goal of the new language implementation. While widely used languages benefit from new implementations simply by exposing ambiguity and holes in the specifications, for Grace there are not yet enough users to justify such an investment.

From the outset, Grace was designed as a language for teaching and research. Looking at its four or five implementations, we see that most of them are interpreters (Kernan is an interpreter built in C#) or compilers to other languages (Minigrace compiles to C and JavaScript). None of these implementations focus on performance. Performance can be a major hurdle for some research questions. For instance, metaobject protocols have been considered impractical for about twenty years, until it could be shown that their performance issues can be solved [Marr et al. 2015]. Similarly, there is an open question about whether gradual typing can be supported efficiently in a practical implementation [Takikawa et al. 2016]. While efficiency was not a goal of the language design [Black et al. 2010], the language should not gratuitously include features that cannot be implemented efficiency. Grace's current design now includes several features (gradual typing, a deeply nested object model, language extension via dialects, inheritance from "fresh objects", method aliasing over inheritance) that are sufficiently novel or complex that their efficient implementation cannot be taken for granted. To validate the language design, then, we need to demonstrate an implementation that achieves performance comparable with contemporary implementations of competing languages. This means the performance goal for our new Grace implementation (called "Moth" [Hopper 1947]) is to reach performance on a level comparable to JavaScript. We aim to reach that goal without requiring a custom VM, a team of 100 engineers, and resources that even large companies struggle to provide.

Grace'18, November, 04, 2018, Boston, MA, USA

2 How do Newspeak and Truffle fit into the mix?

Self [Chambers et al. 1989] and Newspeak [Bracha et al. 2010] had a major influences on Grace's design. Self inspires Grace's foundation upon objects, rather than classes, and its view of computation as requesting messages. Newspeak inspires Grace's ubiquitous nesting used to represent related abstractions, and in particular, to simulate classes with objects.

As such, an early idea was that one may be able to adapt an existing Newspeak implementation with comparably little effort to execute Grace programs. This intuition was confirmed with a one-week effort of getting Grace running on top of a Newspeak implementation simply by translating the AST from an existing Grace parser.

SOMns is a Newspeak implementation on top the Truffle framework and the Graal JIT compiler [Würthinger et al. 2013, 2017]. SOMns is implemented as an abstract-syntax-tree (AST) interpreter, which represents the language semantics directly as AST nodes. Compared to more traditional compilers and bytecode interpreters, an AST interpreter can avoid most compile-time analyses of the input program, and realize all aspects of the language semantics at run time. While this approach drastically simplifies an implementation, doing essentially all the work of checking and evaluating a program at run time means that a naïve AST interpreter will be slow. SOMns leverages the Java Virtual Machine, Graal compiler, and Truffle framework so that its AST interpreter reaches the performance of V8 JavaScript [Marr et al. 2016]. This made SOMns appealing as platform for a new Grace implementation that has performance as a major goal.

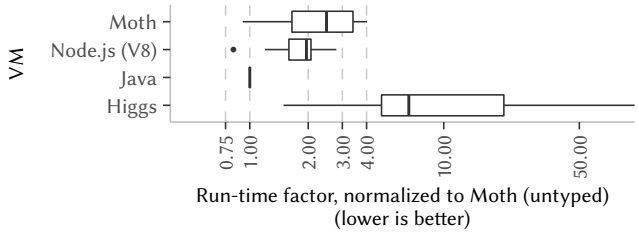

Figure 1. Comparison of Java 1.8, Node.js 10.4, Higgs VM, and Moth. The boxplot depicts the peak-performance results for the Are We Fast Yet benchmarks, each benchmark normalized based on the result for Java. For these benchmarks, Moth is within the performance range of JavaScript, as implemented by Node.js, which means we reached our performance goal.

3 What worked

The promise of SOMns, Truffle, and Graal were great performance at reasonable engineering cost. Looking at the numbers, this is not too far of the mark.

SOMns consists of about 33KLOC of Java and about 2KLOC of Newspeak for its standard library. The biggest difference between Newspeak and Grace's object models is that Newspeak is based on classes, while Grace is based on object literals. The Newspeak specification also defines object literals, but no Newspeak implementation supports them, including SOMns. We extended SOMns to include Newspeak's object literals, with the aim that they could be used for Grace's object literals. Object literal support added about 1.5KLOC of Java.

Moth relies on a third-party parser (Kernan) for parsing Grace code. As a consequence, Moth only needs to process a JSON encoding of the Grace AST, Combined with other smaller pieces of infrastructure, Moth adds about 5KLOC of changes to SOMns, and is therefore a relatively small addition.

Figure 1 shows the performance of Moth compared to Java and Node.js. For the evaluation, we use the Are We Fast Yet benchmarks [Marr et al. 2016], which were designed for such cross-language comparisons. For peak performance, we see that we are 1.2x (min. 0.8x, max. 2.2x) slower than Node.js, and about 2.3x (min. 0.9x, max. 4x) slower than Java.

To put these results into perspective, we also compared Moth with the Higgs VM, a JavaScript VM for research on JIT compilation [Chevalier-Boisvert and Feeley 2015, 2016]. Higgs consists of 21KLOC of D and 9KLOC of JavaScript. While this is small and manageable for a custom VM, Higgs's performance on the Are We Fast Yet benchmarks is not yet in the range of V8 JavaScript.

4 What didn't work

While we reached the main goal that we aimed for, there are a number of issues that still require more engineering and research. Sometimes the language specification does not yet specify concrete semantics. One such example is a fairly basic aspect: numbers. While it is said that Grace must support at least a `Number` type with at least 51 bits of precision, it also says that a full specification of numeric types is yet to be completed. Unfortunately, this is in tension with the choices made in SOMns, which has two distinct numeric types. SOMns `Doubles` are 64bit IEEE 754 floating point numbers, while SOMns `Integers` are arbitrary precision integers with an optimization for 64bit values. At the moment, Moth adopts SOMns' model, which leads to strange effects. The most noticeable is perhaps the proliferation of `.asInteger` method requests which turn a Grace number literal (encoded as a SOMns `Double`), into an SOMns `Integer` to take advantage of integer specific library methods and optimizations.

Other issues are simply an incomplete implementation. For example, Grace's `for` loop uses a range syntax `from..to`. But in Moth, we typically rely on SOMns' `#to:do:` method of `Integer`. The main reason here is that some of these bits simply have not yet been implemented. For most parts, we rely on the SOMns standard library, which means that Moth does not necessarily feel like a proper Grace would.

Other aspects require more research. For instance the support for Grace's dialects is not yet completed. While basic elements work, we do not usually use control flow constructs such as `if (.) then {} else {}` from the dialect. Instead, we request Newspeak's `#ifTrue:ifFalse:` on a boolean object. The main reason here is that dialects are not yet optimized in the same way as SOMns builtin classes that offer such control structures. For best performance, we therefore avoid things like `if/then`. At this point, we do not yet have a good solution to optimize dialects reliably. The use of simple heuristics would mean that the optimization would apply only sometimes, but not always, which could lead to unexpected results.

As mentioned in section 3, we rely on Kernan for parsing. Our initial versions of Moth included a custom parser for Grace. Unfortunately, this turned out to take much more engineering time than we would have wished, and we gave up on it for the moment.

Another issue that remains unsolved is the tension between breaking with SOMns' implementation decisions, and the appeal of being able to adopt SOMns' maintenance changes comparably effortlessly. An ideal Grace implementation would depart from SOMns and chose the simplest possible way to realize Grace semantics. However, this would mean that we have to change SOMns more extensively than we have done so far. While this could lead to an overall smaller code base, and a system that may be easier to understand, it would also mean more effort for maintaining Moth. For instance, keeping up to date with the Truffle library requires work. Similarly, SOMns is actively developed and some of its features such as support for various concurrency models, or the

language server protocol for IDE support, may be relevant in the future. Thus, ideally, any improvements in these parts would be easily adoptable, which would be hampered by more extensive customizations of the SOMns core.

5 Nothing is perfect

Moth is a reasonable approximation of Grace. Newspeak, as expected, was a fairly good foundation for Grace. The similarities outweigh the differences, and SOMns, with a good understanding of how Truffle works in general, enables changes to the language semantics in a more direct way than a source translator or compiler would.

As seen in fig. 1, the peak performance of Moth is 1.2x (min. 0.8x, max. 2.2x) slower than Node.js and therefore reaches our set goals. With such a peak performance, we can evaluate ideas with a good chance of generalizing to state-of-the-art systems. Thus, we are confident that Moth will enable us to pursue a wider range of research questions with Grace.

Acknowledgments

This work is supported by the Royal Society of New Zealand Marsden Fund, and a James Cook Fellowship.

References

Andrew P. Black, Kim B. Bruce, and James Noble. 2010. Panel: designing the next educational programming language. In *SPLASH/OOPSLA Companion*.

Gilad Bracha, Peter von der Ahé, Vassili Bykov, Yaron Kashai, William Maddox, and Eliot Miranda. 2010. Modules as Objects in Newspeak. In *European Conference on Object-Oriented Programming (ECOOP)*. Lecture Notes in Computer Science, Vol. 6183. 405–428. https://doi.org/10.1007/978-3-642-14107-2_20

Craig Chambers, David Ungar, and Elgin Lee. 1989. An Efficient Implementation of SELF a Dynamically-Typed Object-Oriented Language Based on Prototypes. In *Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'89)*. ACM, 49–70. https://doi.org/10.1145/74878.74884

Maxime Chevalier-Boisvert and Marc Feeley. 2015. Simple and Effective Type Check Removal through Lazy Basic Block Versioning. In *29th European Conference on Object-Oriented Programming (ECOOP 2015) (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 37. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 101–123. https://doi.org/10.4230/LIPIcs.ECOOP.2015.101

Maxime Chevalier-Boisvert and Marc Feeley. 2016. Interprocedural Type Specialization of JavaScript Programs Without Type Analysis. In *30th European Conference on Object-Oriented Programming (ECOOP 2016) (LIPIcs)*, Vol. 56. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 7:1–7:24. https://doi.org/10.4230/LIPIcs.ECOOP.2016.7

Grace Hopper. 1947. Log Book With Computer Bug. National Museum of American History.

Stefan Marr, Benoit Daloze, and Hanspeter Mössenböck. 2016. Cross-Language Compiler Benchmarking—Are We Fast Yet?. In *Proceedings of the 12th Symposium on Dynamic Languages (DLS'16)*. ACM, 120–131. https://doi.org/10.1145/2989225.2989232

Stefan Marr, Chris Seaton, and Stéphane Ducasse. 2015. Zero-Overhead Metaprogramming: Reflection and Metaobject Protocols Fast and without Compromises. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '15)*. ACM, 545–554. https://doi.org/10.1145/2737924.2737963

Asumu Takikawa, Daniel Feltey, Ben Greenman, Max S. New, Jan Vitek, and Matthias Felleisen. 2016. Is Sound Gradual Typing Dead?. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'16)*. ACM, 456–468. https://doi.org/10.1145/2837614.2837630

Thomas Würthinger, Christian Wimmer, Christian Humer, Andreas Wöß, Lukas Stadler, Chris Seaton, Gilles Duboscq, Doug Simon, and Matthias Grimmer. 2017. Practical Partial Evaluation for High-performance Dynamic Language Runtimes. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'17)*. ACM, 662–676. https://doi.org/10.1145/3062341.3062381

Thomas Würthinger, Christian Wimmer, Andreas Wöß, Lukas Stadler, Gilles Duboscq, Christian Humer, Gregor Richards, Doug Simon, and Mario 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 (Onward!'13)*. ACM, 187–204. https://doi.org/10.1145/2509578.2509581