Assessing primitives performance on multi-stage execution

First1 Last1
Position1
Department1
Institution1
City1, State1, Country1
first1.last1@inst1.edu

First2 Last2
Position2a
Department2a
Institution2a
City2a, State2a, Country2a
first2.last2@inst2a.com

First3 Last3
Position2b
Department2b
Institution2b
City2b, State2b, Country2b
first2.last2@inst2b.org

Abstract

Most of programming languages uses primitive functions to speed up their execution time ; these primitives usually implements core operations (like addition, substraction,...) or frequently used operations. They are also massively used in the context of dynamic or just-in-time compilation : the compiler then executes this alternative version instead of the standard one to gain performance. Dynamic compilation is slower then static compilation: in this context, one has to set up strategies to reach higher performances. primitives can thus be implemented in the context of just in time compilation.
The Cog VM dynamically compiles oriented-object languages such as Newspeak, Pharo, Squeak: this paper describes the recent implementation of optimised versions of the String comparison primitive at different stages of the code execution and their impact on execution time. [describes briefly the 3 implementations and compares with V8 and Java?]

ACM Reference Format:
First1 Last1, First2 Last2, and First3 Last3. 2018. Assessing primitives performance on multi-stage execution. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Short description of the String class (String, ByteString, WideString)?

2 Terminology

2.1 definitions

- primitive (but already defined in abstract?)
-

Understanding primitive implementation and execution

Conference'17, July 2017, Washington, DC, USA
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2.2 MiscPlugin

Code compiled into bytecode and interpreted
trade-off

- +: easy to code, easy to read, implementation in-image, no need to recompile th vm sources in case of modification
- -: slow

2.3 primitive in Slang

Slang is compiled to C (by usual c compilers gcc/llvm), and the resulting machine code is executed
trade-off

- +: faster execution
- -: costly jumps between Smalltalk and C runtime (trampolines), need to recompile vm sources, little control over outcome (c compilers optimizations)

2.4 primitive in baseline JIT

generate functions as machine code partial implementation (frequent cases are implemented) Sometimes, fallback to less-optimised version

trade-off

- +: faster (faster) execution (no c - Smalltalk runtime jumps), control over the outcome
- -: need to recompile vm sources (trampolines), code readability

2.5 primitive in optimizing JIT

add appendix with code

3 Assessment

All these versions of the primitives have been implemented. Compare performances

3.1 Micro-benchmarks

Results on strings of different lengths (3, 10, 1000)

3.2 Macro-benchmarks

JSON PetitParser XML

4 Related work

- Java: platform dependent
- V8: interpreter compiled by JIT optimised backend as if we were compiling slang using the JIT compiler (ahead of time) no more virtual call
- VisualWorks: baseline JIT but no interpreter

5 Future work and conclusion

1. number of representations
2. common representation

A Appendix

Text of appendix ...