Toward a Dynamic Language Toolkit

Dave Mason
Toronto Metropolitan University

Toronto Metropolitan University

Introduction

- dynamic languages remain important - 20-50 years later
  - Javascript, Python, R, Ruby, Smalltalk, Scheme, Groovy
- Commodity computers have changed dramatically
  - much larger memories
  - multiple computational cores.
- problem sets grow larger

Design Principles for a Modern Runtime

- Large memories
- 64-bit and IEEE-754
- Multi-core and threading
- Fast execution

Zag Smalltalk

- from-scratch implementation
- low-level is implemented in Zig
- goal is to support existing OpenSmalltalk systems
- don't want to rewrite userland!

Key Features

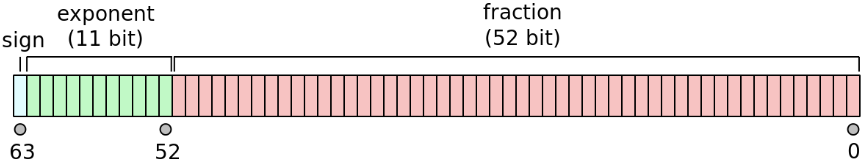
- 64-bit immediates: double FP, 51-bit SmallInteger, true, false, nil, Symbols, Unicode chars, some BlockClosures, heap-object reference
- per-thread copying memory arenas, stacks
- shared non-moving arena
- fast dispatch
- easy code generation from AST
- currently threaded-code execution and stand-alone export
- future JIT

Immediate Values

- 64 bit
- NaN-boxing
- double-floats, 51-bit SmallInteger, Booleans, nil, Unicode characters, Symbols
- room for instances of any type with single 32-bit value
- some kinds of BlockClosures

Immediate Values

- 64 bit
- NaN-boxing
- double-floats, 51-bit SmallInteger, Booleans, nil, Unicode characters, Symbols
- room for instances of any type with single 32-bit value
- some kinds of BlockClosures



Sign+Exp	F	F	F	Type
0000	0000	0000	0000	double +0
0000-7FEF	xxxx	xxxx	xxxx	double (positive)
7FF0	0000	0000	0000	+inf
7FF0-F	xxxx	xxxx	xxxx	NaN (unused)
8000	0000	0000	0000	double -0
8000-FFEF	xxxx	xxxx	xxxx	double (negative)
FFF0	0000	0000	0000	-inf
FFF0	xxxx	xxxx	xxxx	NaN (currently unused)
FFF1	0000	xxxx	xxxx	reserved (tag = unused)
FFF1	0001	xxxx	xxxx	reserved (tag = Object)
FFF1	0002	xxxx	xxxx	reserved (tag = SmallInteger)
FFF1	0003	xxxx	xxxx	reserved (tag = Double)
FFF1	0004	0001	0000	False
FFF1	0005	0010	0001	True
FFF1	0006	0100	0002	UndefinedObject
FFF1	0007	aaxx	xxxx	Symbol
FFF1	0008	xxxx	xxxx	Character
FFF2	xxxx	xxxx	xxxx	immediate thunk
FFF3	xxxx	xxxx	xxxx	niladic constant closure
FFF4	xxxx	xxxx	xxxx	monadic constant closure
FFF5	xxxx	xxxx	xxxx	diadic constant closure
FFF6	xxxx	xxxx	xxxx	self thunk
FFF7	xxxx	xxxx	xxxx	heap object
FFF8-F	xxxx	xxxx	xxxx	SmallInteger

Multi-core support

- only way to speed up applications
- minimal blocking
- computational/mutator threads - typically 1 per core
- I/O threads - one per open I/O port (“file”)
- global collector thread

Threads and Memory Management

- mutator threads
  - copying collector
  - private nursery (includes stack)
  - 2 teen arenas - n copies before promotion
  - when prompted, finds refs to global arena and marks them
  - then can proceed
- I/O threads
  - maintains list of current shared buffers while I/O blocked
- global collector thread
  - non-moving mark/sweep arena
  - periodically does mark
    - marks known shared structures (class table, symbol table, dispatch tables)
    - asks mutators for global roots
    - processes them until all roots have been found
  - then can proceed to sweep

...

... Memory management

- global collector for non-moving mark-&-sweep
- uses linked-freelist heap (similar to Mist)
- cache-line aware
- large objects (e.g. 16Kib) have separately mapped pages (allows mmap of large files) to minimize memory creep

Unified dispatch

- single level of hashing for method dispatch
- each class dispatch table has entry for every method it has been sent - regardless of place in hierarchy
- near-perfect hash using Φ hashing
- standard SPUR/OpenVM optimizations don’t work well in multi-core environments

High performance Inlining

- no special case for `ifTrue`, `whileTrue`, etc.
- references to `self` / `super` code are inlined
- methods with small number of implementations are inlined - rather than heuristic
- prevents creation of many blocks
- provides large compilation units for optimization

Code Generation

- no interpreter, 3 code generation models
- threaded-execution
  - method is sequence of Zig function addresses
  - uses Zig tail-call-elimination - passes pc, sp, hp, thread, context
  - primitives and control implementations
- JIT
  - future
  - driven from the threaded code
  - Continuation-Passing-Style
  - LLVM JIT
  - seamless transitions between native and threaded execution
- stand-alone generator
  - up-front version of the JIT
  - generates Zig code for methods
  - depends on Zig inlining and excellent code generation

Context

- created lazily - only needed if method will send a message
- created on stack
- incomplete objects unless moved to heap

Name	Filled	Description
header	1/2	object header
tpc	call	threaded PC
npc	*	Native PC - CPS
size	promote	# locals+params+2
prevCtxt	create	link to previous context
method	create	method for debug
temps	create	locals set to `Nil`

Supported Control Abstractions

- full closures
- non-local returns
- call-with-current-continuation
- yield
- generators
- coroutines

... all constant time

Results

Running `40 fibonacci`

```
fibonacci
    self <= 2 ifTrue: [ ↑ 1 ].
    ↑ (self - 1) fibonacci + (self - 2) fibonacci
```

...

... Results

version	AArch64	x86-64	RaspPi4[1]	RV64g[1]
Pharo[2]	591ms	676ms	3548ms	106821ms
fibNative:	207ms	168ms	739ms	3442ms
fibObject:	243ms	342ms	1296ms	8147ms
fibComp:	489ms	471ms	2815ms	9217ms
fibThread:	2116ms	2077ms	7301ms	34379ms
fibByte:	14372ms	20780ms	114908ms	3329575ms

- Native is Zig transliteration using the hardware stack and `i64` values
- Object is Zig transliteration using the hardware stack and object values
  - i.e. dynamic typing - including type-checking
  - 17% cost on M1, 100% cost on x86-64
- Comp is hand-compiled, using a separate object stack
  - from the Thread code (what the JIT would produce)
  - 100% cost on M1, 37% cost on x86-64
- Thread is threaded execution - just over 4x slower
- Byte is a byte-coded interpreter - for comparison

[1] thanks to Ken Dickey for RaspPi and RISC-V data
[2] JIT (last 2 on Cuis - 30-50% slower) except RV uses bytecode interpreter

Conclusions

- moving towards high-performance Smalltalk
- some of the ideas may make their ways to other Smalltalks
- planned to also support other languages
- `https://github.com/dvmason/Zag-Smalltalk`