Celebrating Guile 3

FOSDEM 2020, Brussels

Andy Wingo | wingo@igalia.com

wingolog.org | @andywingo
Lessons Learned from Guile, the Ancient & Spry

FOSDEM 2020, Brussels

Andy Wingo | wingo@igalia.com
wingolog.org | @andywingo
spry /spriː/

- adjective: active; lively
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Speed Relative to guile-2.2.6.1-a69b5</th>
</tr>
</thead>
<tbody>
<tr>
<td>pi</td>
<td>0.25x slower</td>
</tr>
<tr>
<td>read1</td>
<td>0.25x slower</td>
</tr>
<tr>
<td>chudnovsky:50:50:50</td>
<td>0.25x slower</td>
</tr>
<tr>
<td>tail</td>
<td>0.50x slower</td>
</tr>
<tr>
<td>string:5000:500:100</td>
<td>0.50x slower</td>
</tr>
<tr>
<td>ram:50</td>
<td>0.50x slower</td>
</tr>
<tr>
<td>div/g</td>
<td>0.50x slower</td>
</tr>
<tr>
<td>early:1</td>
<td>0.50x slower</td>
</tr>
<tr>
<td>sumfp:10000:0:500</td>
<td>0.50x slower</td>
</tr>
<tr>
<td>mpsum:20:10:2:1</td>
<td>0.50x slower</td>
</tr>
<tr>
<td>browse:2000</td>
<td>0.50x slower</td>
</tr>
<tr>
<td>cat:50</td>
<td>0.50x slower</td>
</tr>
<tr>
<td>蚜虫:1000000</td>
<td>0.50x slower</td>
</tr>
<tr>
<td>cpstac:k:40:20:11:1</td>
<td>0.50x slower</td>
</tr>
<tr>
<td>primes:1000:10000</td>
<td>0.50x slower</td>
</tr>
<tr>
<td>xor:10000000:100000</td>
<td>0.50x slower</td>
</tr>
<tr>
<td>quicksort:1000:2500</td>
<td>0.50x slower</td>
</tr>
<tr>
<td>mbbak:2:75:1000</td>
<td>0.50x slower</td>
</tr>
<tr>
<td>paraffins:23:10</td>
<td>0.50x slower</td>
</tr>
<tr>
<td>puzzle:1000</td>
<td>0.50x slower</td>
</tr>
<tr>
<td>fibf:35:0:10</td>
<td>0.50x slower</td>
</tr>
<tr>
<td>mt:65:536:100</td>
<td>0.50x slower</td>
</tr>
<tr>
<td>wc:vinput:1:50</td>
<td>0.50x slower</td>
</tr>
<tr>
<td>dynamic:500</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>nboyer:3:1</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>array1:1000000:0:500</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>di FUNC:60:50:4000</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>di FUNC:1000:100000</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>hqle:13:10</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>scheme:100000</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>nucleic:50</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>sum:10000:200000</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>peak:2000</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>graph:7:3</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>parseq:2500</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>gb:bench:20:1</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>triangl:22:1:50</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>maxfun:11:11:10000</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>shoyer:3:1</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>ray:50</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>lattice:4:4:10</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>simplex:16000000</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>compiler:2000</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>ceform:500</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>maze:20:7:10000</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>fgb:40:5</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>takt:40:20:12:1</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>nfftakt:40:20:12:1</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>ack:3:12:2</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>takt:40:20:11:1</td>
<td>0.60x faster</td>
</tr>
<tr>
<td>mbrot:75:1000</td>
<td>0.60x faster</td>
</tr>
</tbody>
</table>
mini-benchmark: eval

( primitive-eval
  '(let fib ((n 30))
    (if (< n 2)
      n
      (+ (fib (- n 1)) (fib (- n 2))))))

Guile 1.8: primitive-eval written in C
Guile 2.0+: primitive-eval in Scheme
macro-benchmark:
guix

```
guix build libreoffice ghc-pandoc guix \  
   --dry-run --derivation

7% faster
```

```
guix system build config.scm \  
   --dry-run --derivation

10% faster
```
spry /sprɪ/  
noun adjective: (especially of an old person) active; lively
guile is ancient

2010: Rust
2009: Go
2007: Clojure
1995: Ruby
1995: PHP
1995: JavaScript

1993: Guile (3³ years before 3.0!)
built from ancient parts

1991: Python
1990: Haskell

1990: SCM
1989: Bash
1988: Tcl

1988: SIOD
written in an ancient language

1987: Perl
1984: C++
1975: Scheme
1972: C
1958: Lisp
1958: Algol
1954: Fortran
1958: Lisp
1930s: λ-calculus (3^4 years ago!)
Men make their own history, but they do not make it as they please; they do not make it under self-selected circumstances, but under circumstances existing already, given and transmitted from the past.

The tradition of all dead generations weighs like a nightmare on the brains of the living. [...]

_Eighteenth Brumaire of Louis Bonaparte_, Marx, 1852
Languages evolve; how to remain **minimal**?

Dialectic opposites

- world and guile
- stable and active
- ...

Lessons learned from inside Hegel’s motor of history
hill-climbing is insufficient

Ex: Guile 1.8; Extend vs Embed
users stay unless pushed away

Inertial factor: interface

❖ Source (API)
❖ Binary (ABI)
❖ Embedding (API)
❖ CLI
❖ ...

Ex: Python 3; local-eval; R6RS syntax; set!, set-car!
you can’t keep all users

What users say: don’t change or remove existing behavior

But: sometimes losing users is OK. Hard to know when, though

No change at all == death

Natural result of hill-climbing

Ex: psyntax; BDW-GC mark & finalize; compile-time; Unicode / locales
every interface is a cost

Guile binary ABI: libguile.so; compiled Scheme files

Make compatibility easier: **minimize** interface

Ex: `scm_sym_unquote`, GOOPS, Go, Guix
parallel installs for the win

Highly effective pattern for change

libguile-2.0.so

libguile-3.0.so

https://ometer.com/parallel.html

Changed ABI is new ABI; it should have a new name

Ex: make-struct/no-tail, GUILE_PKG([2.2]), libtool
Deprecation facilitates migration.

```
__attribute__((__deprecated__))

(issue-deprecation-warning
 "(ice-9 mapping) is deprecated."
 "Use srfi-69 or rnrs hash tables instead.

scm_c_issue_deprecation_warning
 ("Arbiters are deprecated."
 "Use mutexes or atomic variables instead.

begin-deprecated,
SCM_ENABLE_DEPRECATED
```
Replace, Deprecate, **Remove**

All change is possible; question is only length of deprecation period

Applies to all interfaces

Guile deprecation period generally one stable series

**Ex:** `scm_t_uint8`; `make-struct`

Foreign objects; uniform vectors
change produces a new stable point

Stability within series: only additions

Corollary: dependencies must be at least as stable as you!

❖ for your definition of stable
❖ social norms help (GNU, semver)

Ex: libtool; unistring; gnulib
who can crank the motor of history?

All libraries define languages
Allow user to evolve the language

❖ User functionality: modules (Guix)
❖ User syntax: macros (yay Scheme)

Guile 1.8 perf created tension
❖ incorporate code into Guile
❖ large C interface “for speed”

Compiler removed pressure on C ABI
Empowered users need less from you
contributions

From maintenance point of view, all interface is legacy

Guile: Sometimes OK to accept user modules when they are more stable than Guile

In-tree users keep you honest

Ex: SSAX, fibers, SRFI
Memory management is an ongoing thorn

Local maximum: Boehm-Demers-Weiser conservative collector

How to get to precise, generational GC?

Not just Guile; e.g. CPython __del__
future

We are here: stability

And then?

- Parallel-installability for source languages: `#lang`
- Sediment idioms from Racket to evolve Guile user base

Remove myself from “holding the crank”
dialectic, boogie woogie

https://gnu.org/s/guile
https://wingolog.org/
guile on freenode
@andywingo
wingo@igalia.com
Happy hacking!