<2022-11-18 Fr> Dr. Arne Babenhauserheide / draketo.de

Is Guile fast?

The effective speed of GNU Guile depends on the task. It can be more than factor 10 faster than Python (i.e. for math with huge numbers) or slower (string manipulation from files which in Python is directly handed off to C-code).

I once resolved performance problems by just translating a Python program into a Guile one using exact numbers. The performance of Guile for huge numbers is awesome!

Contents

1 Compared to other Scheme implementations

In general, Guile is among the faster Schemes, but performance of Schemes varies strongly by task. A performance-comparison of different Schemes over a wide variety of tasks is available in the r7rs-benchmarks by ecraven.

Using geometric mean to obtain one number shows that Guile is roughly factor 1.5 slower than Racket, but as said, this varies by task and approach: from factor 23 faster (string) to factor 40 slower (ctak) with most tests between 3x faster and 4x slower, and this changes a lot when choosing a different approach for the same task.

To get an idea of the spread of the results, I created a plot with Geometric mean and geometric standard deviation from the r7rs-benchmarks data from January 2026:

Mind the logarithmic y-axis, check the numbers on the plot if you're uncertain.

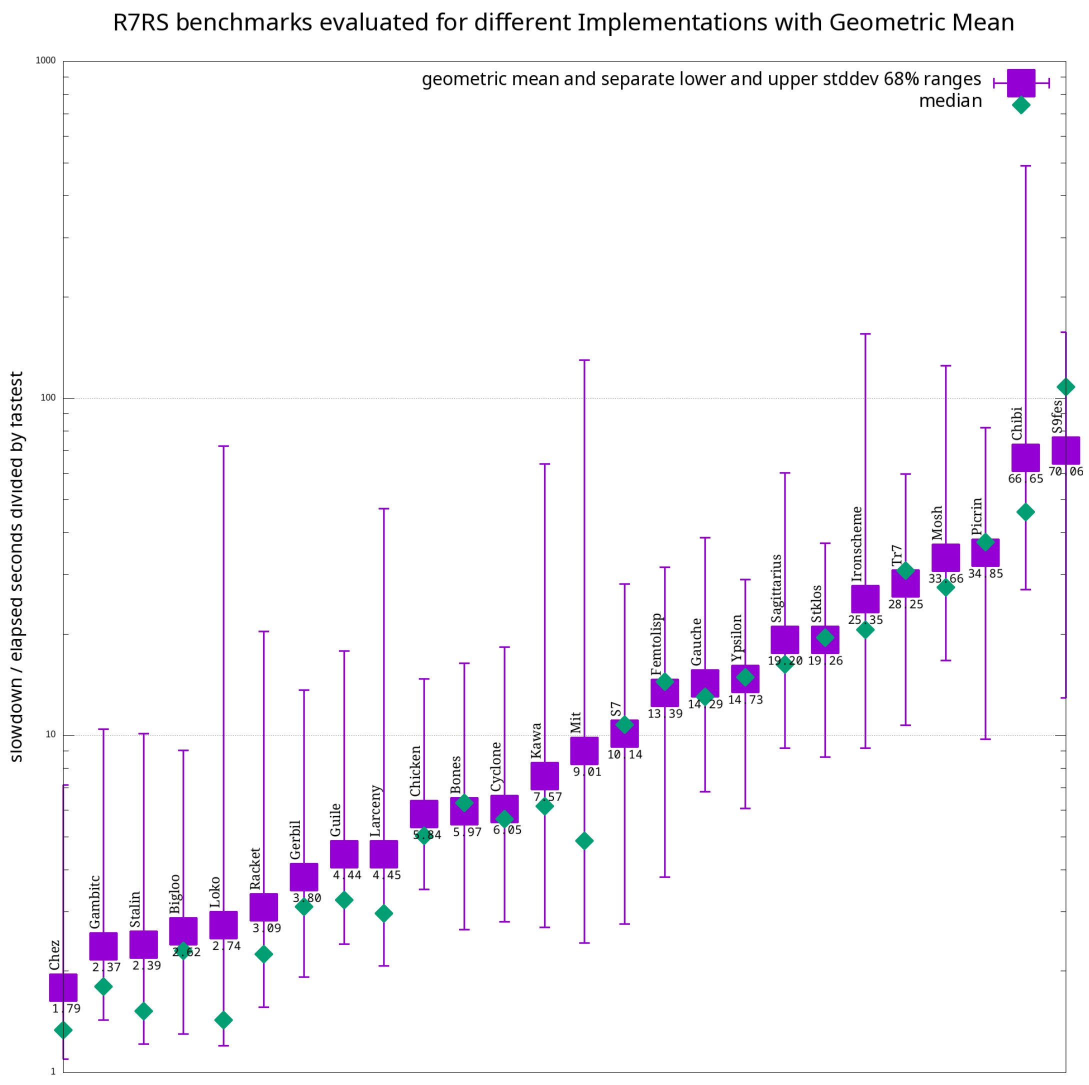


Figure 1: Plot of the geometric mean of the slowdown of benchmarks per Scheme implementation, along with the geometric standard deviation, separated into upper and lower. Chez is at 1.79, Guile is at 4.44.

2 Compared to other programming languages

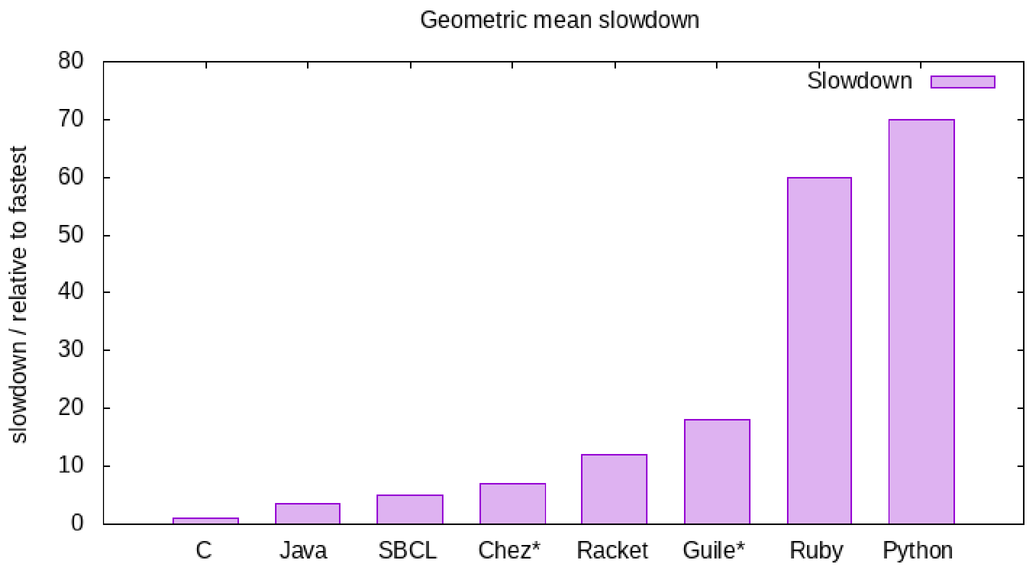
To get an idea how this compares to other programming languages, you can have a look at the results from the benchmarksgame which includes Racket: In the central 50% of tests (the inter-quartile-range, IQR) Racket is on average factor 5-10 slower than C (though with code golf and unsafe ops it got down to 2x C-without-assembler in the spectral norm). In geometric mean racket is about factor 12 slower than C.

Guile is factor 1.44 slower than Racket, in geometric mean, so it could be factor 18 slower than C, while Ruby and Python are about factor 60 to 75 slower than C. Compared to other Lisps, Racket is factor 2.5 slower than SBCL, so Guile is likely about factor 3.5 slower than SBCL (the fastest commonlisp). Compared to Java it is roughly factor 5 slower.

Table 1: Approximate geometric mean slowdown as of 2026-01, taken by eye from the benchmarksgame. *Guile* and Chez* only via their relative speed to Racket.*

	Slowdown
C	1.1
Java	3.5
SBCL	5
Chez*	7
Racket	12
Guile*	18
Ruby	60
Python	70

To put that into an image:



Keep in mind that if people put more work into optimizing the Racket code in the benchmarksgame, this would likely **look** faster.
And note that part of the speed of C here is that people use hand-written vector instructions, so other languages may be closer to C than the numbers suggest.

3 Getting maximum performance

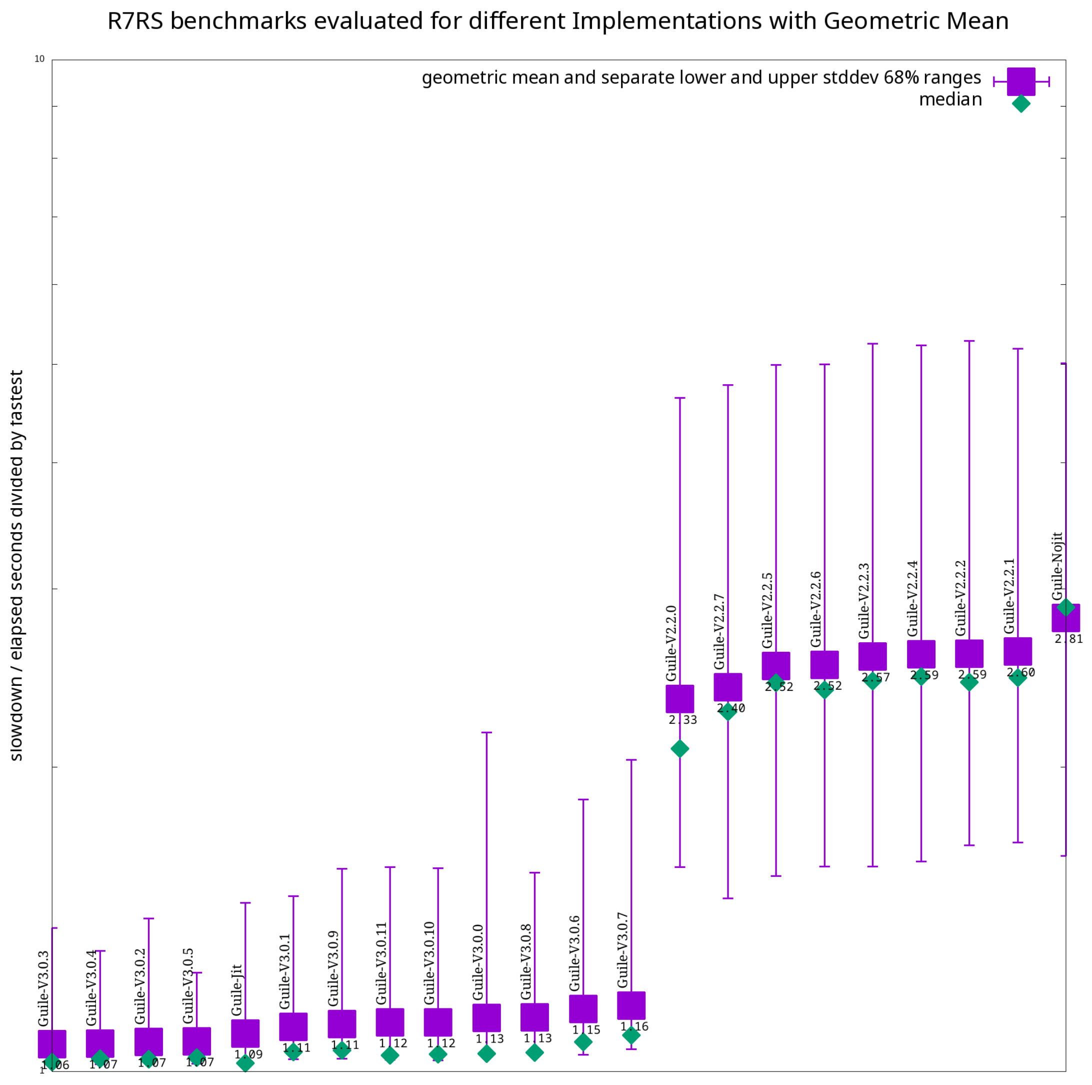
If you need maximum performance, you can **move performance-critical parts to C** and access them via FFI, use **existing bindings** like guile-ffi-cblas or create **low-level helpers** in C.

You can find an example for the latter is in guile-fibers: a library which adds lightweight, multi-process user-level threads to Guile. It implements the performance critical functionality in epoll.c and provides its primitives via `init_fibers_epoll` which it then uses from epoll.scm.

But before you go for C, look at the best-practices article Optimizing Guile Scheme by dthompson, the author of the Chickadee Game Toolkit. You may be already able to improve the speed of your vanilla Scheme, and if that suffices, you avoid the hurdles that come with needing to deploy compiled code.

4 Benchmarking different Guile Versions with the R7RS Benchmarks

To test changes in speed between different guile Versions, I use the R7RS benchmarks by ecraven as well as my own evaluation to get the geometric mean of the slowdown compared to the fastest:



These are no representative statistics! The geometric mean is calculated over different tests, each run only once. The calculation does not propagate the uncertainty of individual test runs!

(This should not be interpreted as recommendation to always force the JIT. In normal operation letting Guile decide when to JIT-compile should provide a better tradeoff than basing such decisions on synthetic benchmark results)

In short: **Guile 3 is roughly factor 2 faster than Guile 2**, and that's due to the just in time compilation.

How to reproduce:

```
CURDIR="$(realpath .)"
TMPDIR="$(mktemp -d "/tmp/guile-fast-XXXXXXXX")"
cd "$TMPDIR" || exit 2
hg clone https://hg.sr.ht/~arnebab/wisp # needs https://mercurial-scm.org
```

```
git clone https://codeberg.org/guile/guile.git # needs https://git-scm.com
git clone https://github.com/ecraven/r7rs-benchmarks
cd guile && git checkout main # replace main with the revision to test
guix environment guile # opens a shell, needs to run on https://guix.gnu.org
guix shell gperf # needed to build guile from shell
./autogen.sh && ./configure --prefix=$HOME/.local && make -j6

# compare Guile v2 and v3
VERSIONS="v2.2.0 v2.2.1 v2.2.2 v2.2.3 v2.2.4 v2.2.5 v2.2.6 v2.2.7 \
  v3.0.0 v3.0.1 v3.0.2 v3.0.3 v3.0.4 v3.0.5 v3.0.6 v3.0.7 v3.0.8 \
  v3.0.9 v3.0.10 v3.0.11"
for VERSION in $VERSIONS; do
  git checkout $VERSION && make clean
  autoreconf -i && ./configure && make -j6
  cd ../r7rs-benchmarks
  rm -results.Guile
  guix shell guile -- bash -c \
    'GUILD=../guile/meta/guild GUILE=../guile/meta/guile ./bench guile all'
  sed -i s/guile-/guile-${VERSION}-/g results.Guile
  sed -i s/guile,/guile-${VERSION},/g results.Guile
  mv results.Guile results.Guile${VERSION}
  cd -
done
cd ../r7rs-benchmarks

# benchmark guile 3 with forced jit
rm results.Guile
VERSION=jit
guix shell guile -- bash -c \
  'GUILE_JIT_THRESHOLD=0 \
   GUILD=../guile/meta/guild \
   GUILE=../guile/meta/guile \
     ./bench guile all'
sed -i s/guile-/guile-${VERSION}-/g results.Guile
sed -i s/guile,/guile-${VERSION},/g results.Guile
mv results.Guile results.Guile${VERSION}

# benchmark guile 3 without jit
VERSION=nojit
guix shell guile -- bash -c \
  'GUILE_JIT_THRESHOLD=-1 \
   GUILD=../guile/meta/guild \
   GUILE=../guile/meta/guile \
     ./bench guile all'
```

```
sed -i s/guile-/guile-${VERSION}-/g results.Guile
sed -i s/guile,/guile-${VERSION},/g results.Guile
mv results.Guile results.Guile${VERSION}

# benchmark guile 3 with parameters that better match to production
VERSION=opt
echo '(define arithmetic-shift ash)' >> src/Guile3-prelude.scm
guix shell guile -- bash -c \
   GUILD=../guile/meta/guild \
   GUILE=../guile/meta/guile \
   GC_INITIAL_HEAP_SIZE=100000000 \
     ./bench guile all'
sed -i s/guile-/guile-${VERSION}-/g results.Guile
sed -i s/guile,/guile-${VERSION},/g results.Guile
mv results.Guile results.Guile${VERSION}

# collect the results
grep -a -h CSVLINE results.Guile* | sed 's,+!CSVLINE!+,,' >  /tmp/all.csv
cd ../wisp/examples

# create the benchmark graph
./graph-r7rs-benchmarks.w /tmp/all.csv \
  $(for i in $VERSIONS jit nojit; do echo guile-$i; done)
exit; exit; cp ../wisp/examples/r7rs-plot.png "$CURDIR"/
```

You cannot copy this code into a shell, because `guix shell` creates a new shell. But you can paste it into a `*shell*` buffer in Emacs.

This code will run for many hours.

Guile Jit and Guile Nojit are runs with just-in-time compilation forced (Jit) or disabled (Nojit). See GUILE_JIT_THRESHOLD in the handbook.

5 Summary

Guile is among the fasters Scheme implementations: about factor 1.44 slower than Racket, a bit faster than Chicken.

Compared to other programming languages, it is roughly factor 18 slower than C, factor 4 slower than Java or SBCL, factor 3-4 faster than Ruby or Python.

But benchmarks are treacherous: depending on the task, these relations can differ, many even reverse (except for the ones that compare to C with inline SIMD). I've done my best here to show their uncertainties, but general evaluations can never fully match your specific requirements, so do test for yourself, and check what you can optimize.

When your own Guile code seems slow, have a look at the best-practices article Optimizing Guile Scheme by dthompson, the author of the Chickadee Game Toolkit.

If you came here to answer the question “should I use Guile?”, then the article 10 ways GNU Guile is 10x better may give better answers.