Some observations while writing a collection of optimized PRNGs in JS

Eris TSX • 4 Feb 2024

Perf. optimization

- JS engines don’t like unsigned integers. Unsigned integers are usually slower than signed.
- Things that are fast in JIT engines may be slow in jitless, **and vice versa**.
- Functional JavaScript is a slow as hell. Ramda is slow as hell. In JITs, in jitless, it’s always super S-L-O-O-O-W. *All-my-homies-hate-Ramda.jpeg*
- How to write optimized JS **for JITs**:
    1. Write it in asm.js.
    2. Validate.
    3. Fix warnings.
    4. Remove the `use asm` pragma.
    5. Move the code around as you want.
    6. But **never** erase the type hints.
    7. asm.js is usually slower than **the same** code without the `use asm` pragma.
- How to write optimized JS for **jitless engines**:
    1. As less **JavaScript** operations as possible. Not the instructions a corresponding C code could be compiled into with optimizing compiler.
    2. Also take in account algorithms defined in the spec for each operation.
    3. Don’t assume anything like *it’s easy to prove the invariants and replace the code (even interpreted bytecode) with faster instructions*.
    4. Certain things that are slow in JITs (like generators and other way-too-high-level sugar) are usually fast in jitless engines.
    5. asm.js-like code that is fast in JITs may be slow in jitless.
    6. **Don’t do this** unless you absolutely need this.
- *WASM is fast*, but the interop cost is too high, so calling WASM functions from JS side can be **much slower** than writing the same code in plain JS and calling it with no interop overhead.

- SpiderMonkey is slower than V8 and JSCore.
- Jitless SpiderMonkey is slower than jitless V8 and QuickJS.
- Jitless V8 is the fastest of jitless engines.
- QuickJS is quick (but still slower than jitless V8).
- `crypto.getRandomValues()` is not slow (when cached).
- Closures and closure-based objects are slower than class-based objects.
    - Even if methods are defined as arrow functions (i.e. closures).
- Oh, bigints are slooooow (as expected).

Javascripters

- Javascripters don't know how to write optimized code. Well, with some notable exceptions.
- Javascripters don't know how to write *good* PRNGs. And introduce questionable bullshit like, ahem, Gbson's UHEPRNG.
- Sometimes javascripters know how to write optimized code, but don't know how to construct high-quality RNGs. And introduce things like Alea.

Alea

Alea is really fast. In **jitless** engines *(in JITs it's even slower than [optimized] MT19937)*. It's based on Marsaglia's MWC algorithms, but the state is too smol, the constants are too unclear, and the hash it uses to seed the state is *"Oh dear what is `mash()` even doing…."* according to Appleby. Well, I could test it with TestU01 and PractRand, but I'm too lazy to do this.