I was experimenting with snark proof generation inside a browser for mobile phones and made a benchmark: I've tested a simple circuit (5 rounds of pedersen commitment) written on Circom and ported to browser using browserify, and the same circuit written on Bellman and compiled to WebAssembly; I also compared them to their native counterparts.

MacBook Core-i7 @ 2.6Ghz circom (node.js): 53685ms circom (chrome): 53266ms circom (websnark): 1408ms bellman (webassembly): 1151ms bellman (rust, 1 core): 235ms bellman (rust, 8 cores): 85ms

iPhone XS bellman (webassembly): 1126ms

As we can see Bellman compiled to wasm is already 50x faster and still has a room for more than 10x improvement as WebAssembly execution speed should be pretty close to native speeds when everything is done right. The full support for wasm compilation was recently merged into Matter labs fork of Bellman, and it was uploaded to crates.io and can now be incuded as bellman_ce

, you can see wasm-bellman test repo for example usage. On modern phones the execution speeds are similar to laptop speeds, so in the current state wasm compilation seems to be good enough for many simple snark proofs on mobile.

The main bottleneck is that when compiled to wasm Bellman is single threaded like Circom. If we take advantage of multiple CPU cores (most modern phones have 8), we can speed this up significantly. WebAssembly in Chrome already supports multiple threads with shared memory as experimental feature, so we are waiting when the rust compiler will support it (alternatively it can be done via importing WebWorker api). Additionally, the browser profiler [1], [2] shows that most CPU time is spent in __multi3 that is called from mul_assign followed by internals of mul_assign itself, so maybe there are some inefficiencies in int64 multiplication implementation on wasm. When compiled to native mobile app, Bellman can already take advantage of all the cores and calculate proofs even faster than in browser. @shamatar from Matter labs did a few tests on that and got some impressive results.

We currently work on Circom -> Bellman export so that we can combine the ease of use of Circom language and performance and portability of Bellman. The goal is to make a toolchain that allows you to write snarks in Circom and get super fast .wasm + js bindings as compilation result, hiding the complexity of Bellman under the hood.

Benchmark details

The circuit verifies that a hash corresponds to a private 256 bit preimage after taking 5 rounds of pedersen commitment. Resulting circuits on Circom and Bellman have similar number of constraints:

3459 bellman 3540 circom

Circom has a little bit more partially because it calculates hash in 256 bit field compared to 254 bit in Bellman, but the difference is negligible.

The tests were done on a 5 year old laptop with 2.6GHz Core i7 CPU, and an iPhone XS.

Circom source: https://github.com/poma/circom-pedersen-benchmark

Native Bellman source: https://github.com/poma/bellman_pedersen

WebAssembly Bellman source: https://github.com/poma/wasm-bellman

In this last repo you can see an example of how to build your snark that will work within a browser.

See also: @kobigurk benchmark and demo of Bellman in wasm

update:

added websnark benchmark for the same circuit