I've just released a new version of Distaff VM (v0.5) in which I completely overhauled the internal representation of programs. The new structure is described here and it enables a lot of new exciting capabilities. Main new features in this release are:

1. All practical limitations on conditional execution (if-then-(else) statements) have been removed. Now, for every level of nested conditional logic, VM needs to allocate just one extra register. This means that complexity of programs grows linearly in the number of nested conditional statements (as opposed to exponential growth in the previous version).

2. The VM now supported bounded counter-based loops (repeat statements) and unbounded condition-controlled loops (while statements). While loops require one extra register per level of nesting, but the complexity still grows linearly in the level of loop nesting.

3. It is now possible to selectively reveal sub-sections of secret programs. This could be useful for secret smart contracts with public pre/post-conditions.

4. A few new useful instructions have been added as well. A couple of examples:

a. and

and or

boolean operations for binary values.

b. isodd

operation to determine if a value is odd or even.

The entire assembly language for Distaff VM is described here.

With the above improvements, it is now possible to write pretty sophisticated programs for Distaff VM. For example, the program below, reads a secret input and computes a Collatz sequence starting from the specified value. The output of the program is the number of steps it takes to get from the starting value to 1.

begin pad read dup push.1 ne while.true swap push.1 add swap dup isodd.128 if.true push.3 mul push.1 add else push.2 div end dup push.1 ne end swap end

(a runnable version of this example is here).

By executing the above program you can prove that you know a number which results in a Collatz sequence of a given length, without revealing this number.

Impact on performance

Adding support for all these new features (especially for new conditional execution and unbounded loops) resulted in considerable complexity. This led to about 30% increase in proof generation time. But, in my opinion, the trade-off is absolutely worth it.

Here are very informal benchmarks run on Intel Core i5-7300U @ 2.60GHz (single thread) with 8 GB of RAM:

Operation count

Execution time

Execution RAM

Verification time

Proof Size

210

350 ms

negligible

2 ms

80 KB

214

4.5 sec

250 MB

3 ms

132 KB

218

1.3 min

5.5 GB

3 ms

193 KB

Another way to think about performance is that the VM is currently running at ~4 KHz on a single CPU core. With some optimization and multi-thread execution this can be increases considerably, and could approach 40 KHz on an 8 core machine.

Moving VM logic into FPGA could potentially get to 1 MHz speeds, and ASICs could take it beyond that (maybe even to 100 MHz).

Turing-completeness

As far as I can tell, the only thing missing for functional Turing-completeness is random access memory. I have pretty good ideas of how to implement it, and in my rough estimate it will slow down the VM by another 20% or so. But, this slow down will be realized only by programs which need RAM. Programs which can get away with using only stack would not incur any performance penalties.