Benchmarking PlonK Proving System

Lehigh University

Tal Derei and Ben Aulenbach

December 2022

1 Abstract

PlonK (Permutations over Lagrange-bases for Oecumenical Non-Interactive Arguments of Knowledge) [1] is a zero-knowledge proving system that allows a prover to convince a verifier they possess certain information without revealing that information. The objective of this paper is to analyze the performance of PlonK and identify areas for further research. All benchmarks are collected using Aztec's Barretenberg cryptographic library on the CPU [2].

Groth16 is another zero-knowledge proving scheme that requires a circuit-specific trusted setup during the preprocessing phase. In contrast, PlonK also requires a trusted setup in its proof construction, but it is universal and not circuit-dependent. This means it can be used to create proofs for any computation, rather than being specific to a particular circuit. Aztec's "Ignition" trusted setup transcripts support circuits with up to approximately 2^{26} – 2^{27} constraints, or 100.8M constraints [3].

2 Cloud Computing Environment

We measure the relative speed up in performance across the following bare-metal machine instantiated on **Oracle Cloud**: 128-core Intel(R) Xeon(R) Platinum 8358 CPU @ 2.60GHz Base and 3.40 GHz Turbo, 1024 GB DDR4 DRAM, 128 GB SSD.

3 Experimental Results

To measure performance, we used a combination of the Oracle Cloud Infrastructure (OCI) monitoring service and real-time process monitoring tools such as `htop` and `nvtop`. We calculated all performance metrics using 99th percentile statistics and set the maximum constraint size for the circuits at 2^{26} gates.

We conducted benchmarks that fall into three distinct workloads: Arithmetic, Polynomial, and Prover. The **Arithmetic** benchmarks focused on measuring the performance of finite field arithmetic and elliptic curve operations over the BN254 elliptic curve. The **Polynomial** benchmarks tested the execution time and memory efficiency of the multi-scalar multiplication implementation, which uses techniques such as Montgomery's batch inversion trick [4], and fast-fourier transform. The **Prover** benchmarks measured the overall performance of the entire proof generation process. We conducted these benchmarks in three different modes. In all of the benchmarks, multithreading using the OpenMP API was enabled by default:

→ **Normal Mode:** disabled assembly instructions and compiler optimizations.

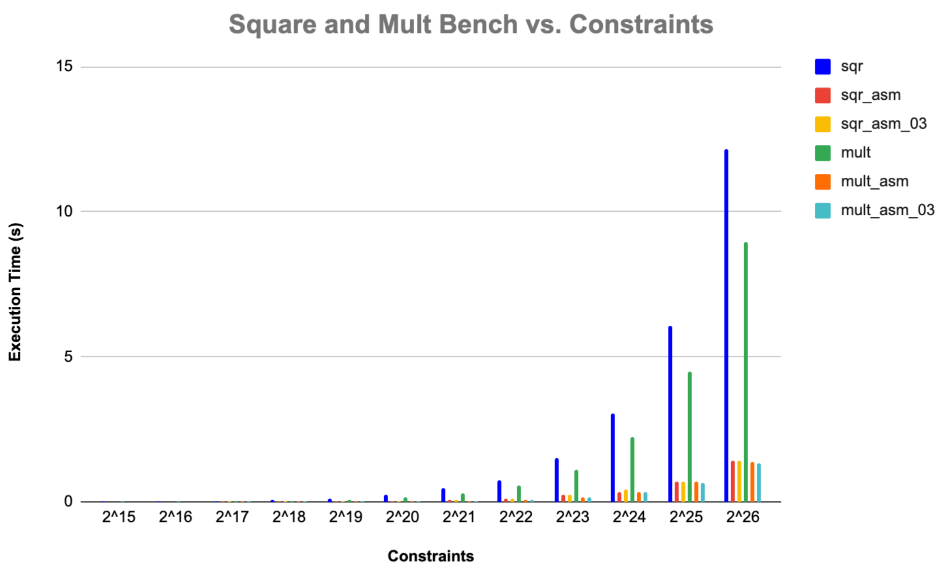
→ **Assembly Instructions (ASM):** enabled assembly instructions by setting the BMI2 macro to detect the BMI2 hardware instruction set.

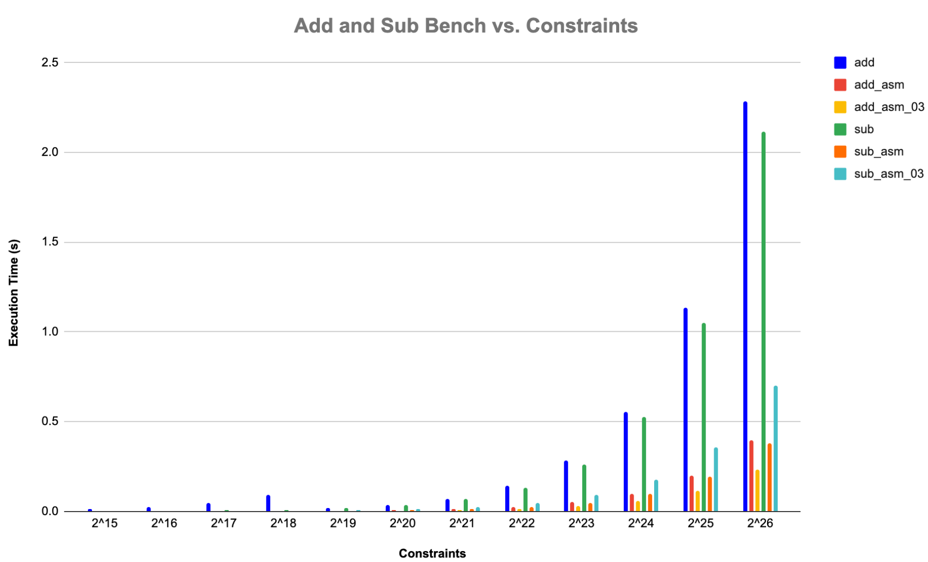
→ **ASM with Compiler Optimizations (03)**: enabled assembly instructions and clang compiler optimizations.

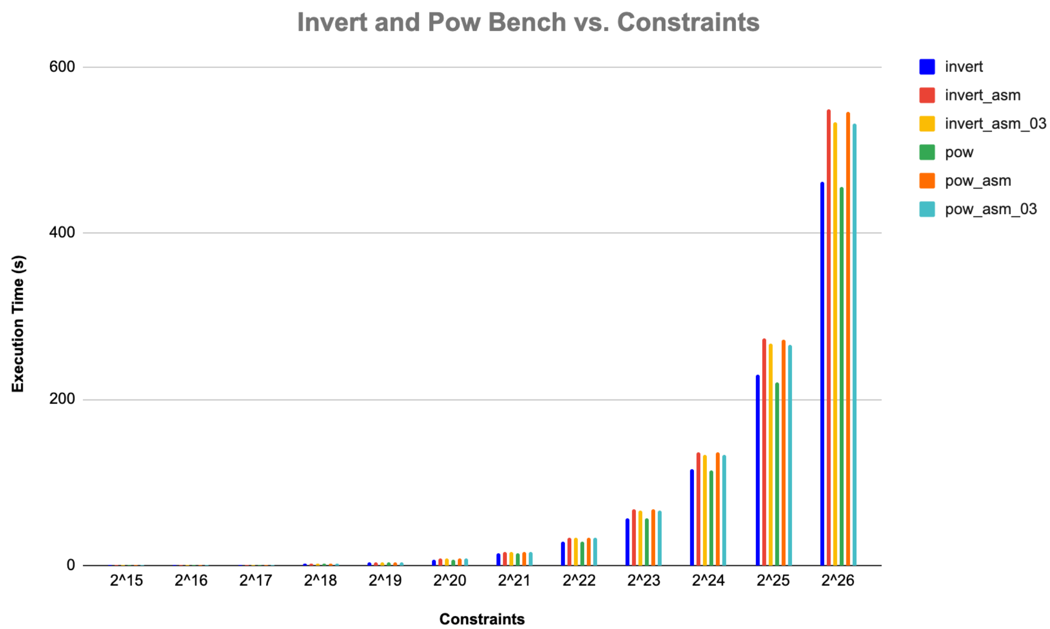
3.1 Arithmetic

The following highlights the performance of finite field arithmetic and elliptic curve operations over BN254. The Squaring and Montgomery multiplication operations with ASM and compiler optimizations significantly reduced the execution time by 88.3% and 85.2%, respectively to sub-2 seconds for 2^{26} constraints. The Invert and Pow operations with ASM and compiler optimizations increased the execution times by 15.49% and 16.87% respectively to ~530s for 2^{26} constraints. Compared to squaring and multiplication, invert and pow operations took ~350–400x longer when executing 2^{26} constraints. Addition and subtraction arithmetic is the most trivial of the calculations.

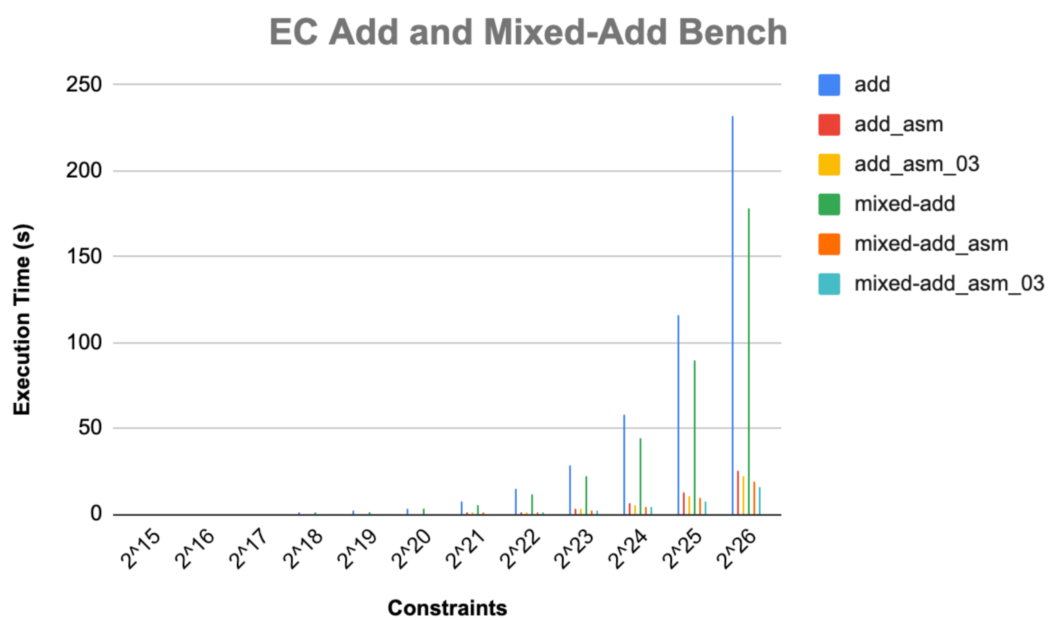
3.1.1 Finite Field Arithmetic







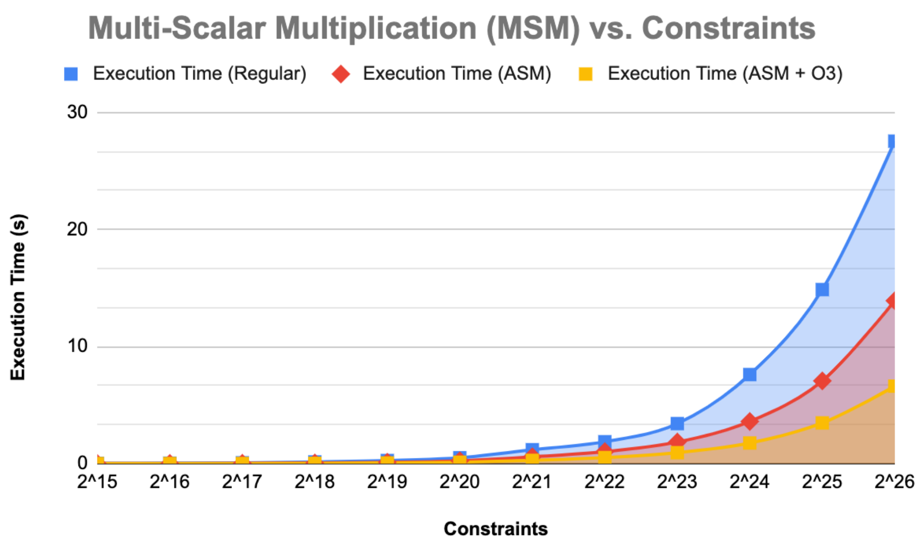
3.1.2 Elliptic Curve Operations Field Arithmetic

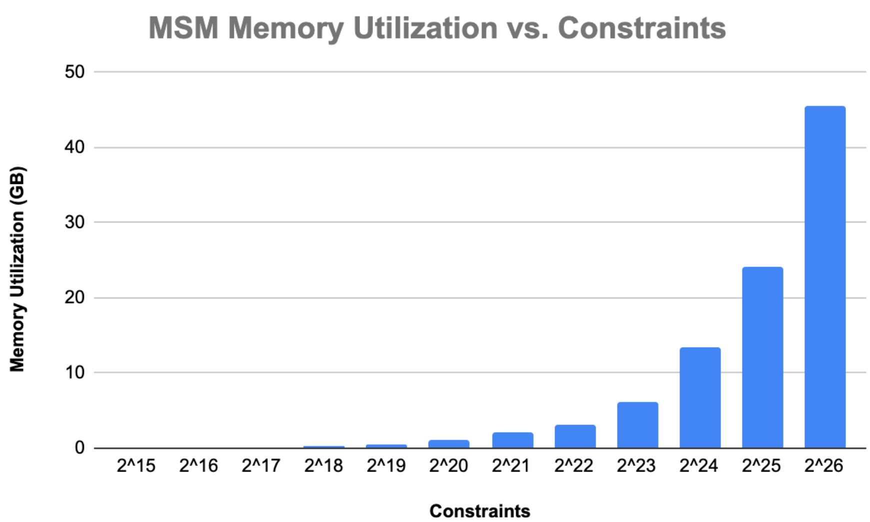


Optimized elliptic curve addition and mixed addition operations took approximately 22s and 16s respectively for executing 2^{26} constraints, much longer compared to its finite field counterpart.

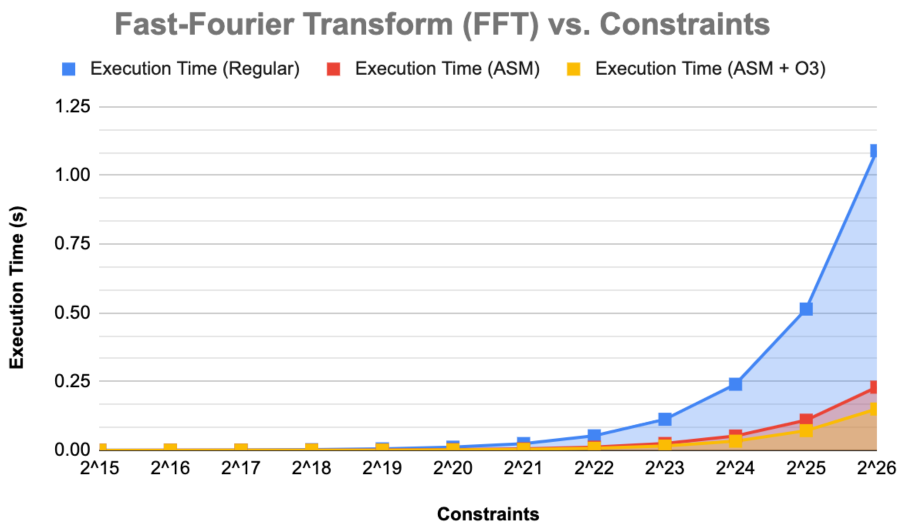
3.2 Polynomial

The following highlights the multi-scalar multiplication (MSM) and faster-fourier transform (FFT).





The figures exemplify that execution time and memory utilization grow exponentially with respect to the number of constraints in the program. Memory utilization includes the initialization phase to generate the scalars and load the parameters into system memory for the prover to access. At 2^{26} constraints, the multi-scalar multiplication takes ~6.7 seconds with ASM + O3, consuming 45.5 GB of memory.



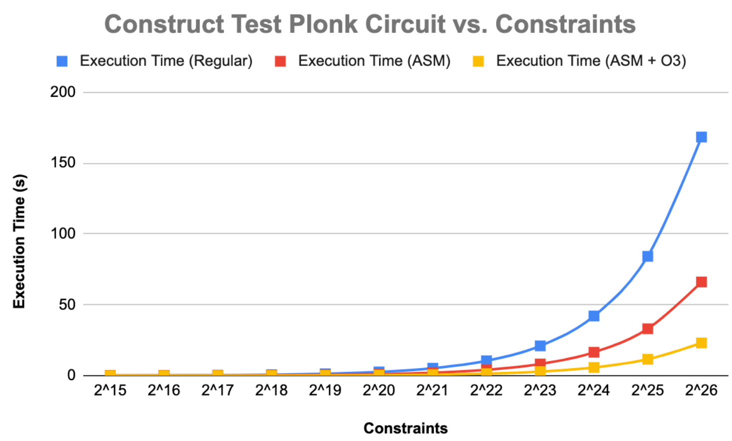
The FFTs also grow exponentially with respect to the number of constraints in the program. At 2^{26} constraints, the fast-fourier transform takes ~0.15 seconds with ASM + O3. The FFT memory utilization is negligible compared to the MSM memory utilization.

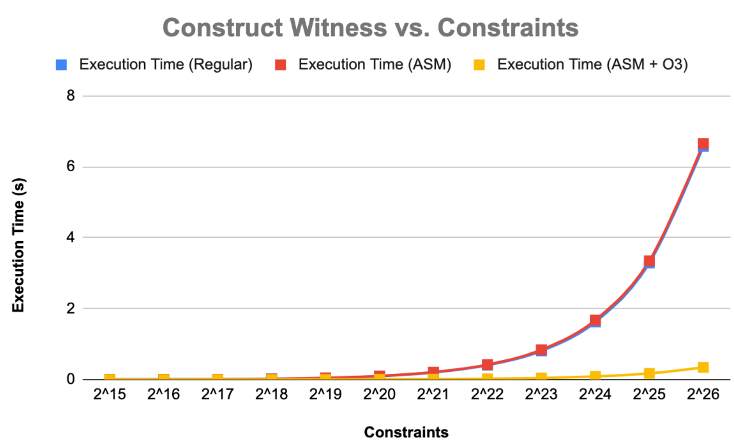
3.3 Prover

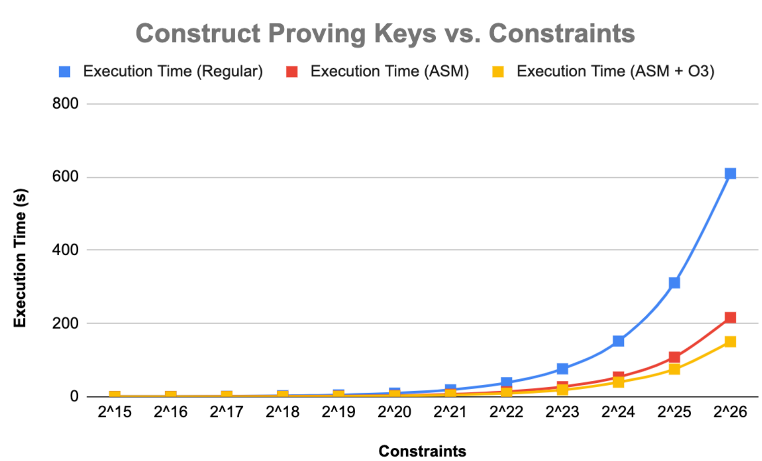
The prover workload is split into 6 tasks:

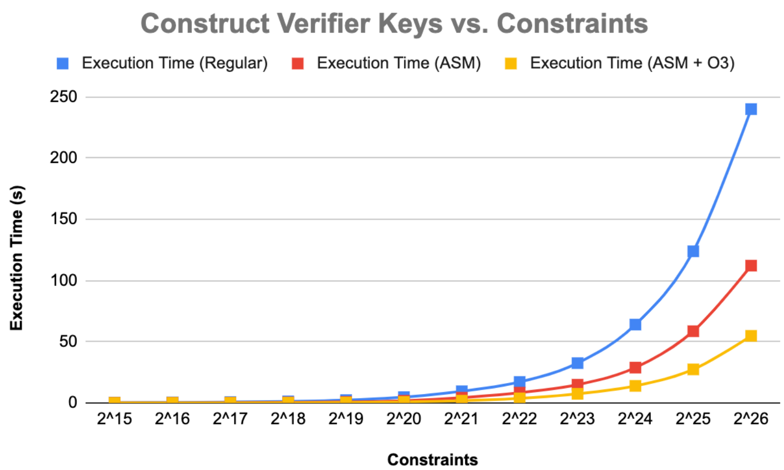
1. Generating the test plonk circuit with a specified maximum number of gates
2. Calculating witness polynomials
3. Computing the proving key, including q_l, q_r, and sigma polynomials
4. Generating the verifier key
5. Creating proofs
6. Verifying proofs

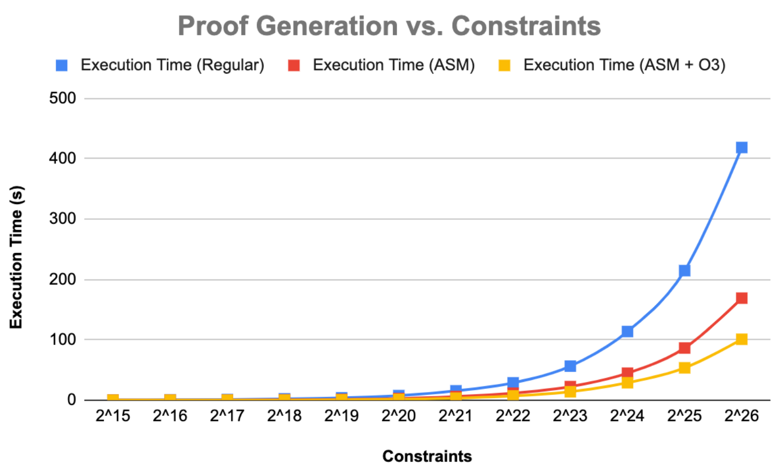
The following charts highlight that for 2^{26} constraints, proof generation took ~100s with 255 GB of system memory. The verification was constant at approximately 2 ms, and generating prover keys took about 3x longer than generating verifier keys.

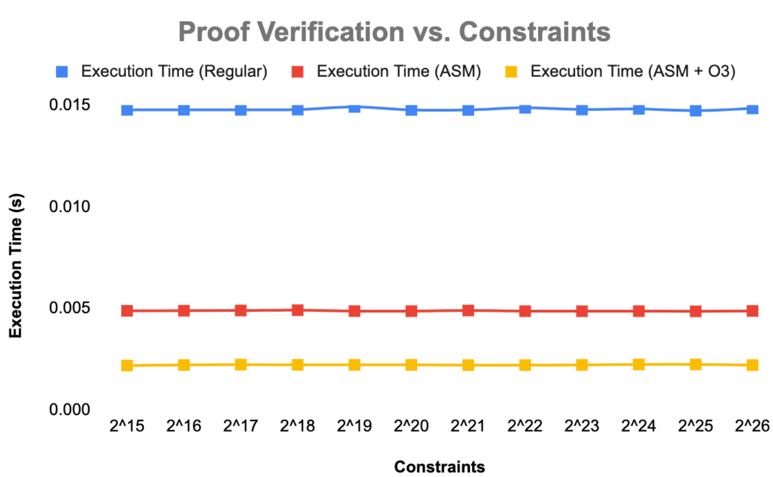


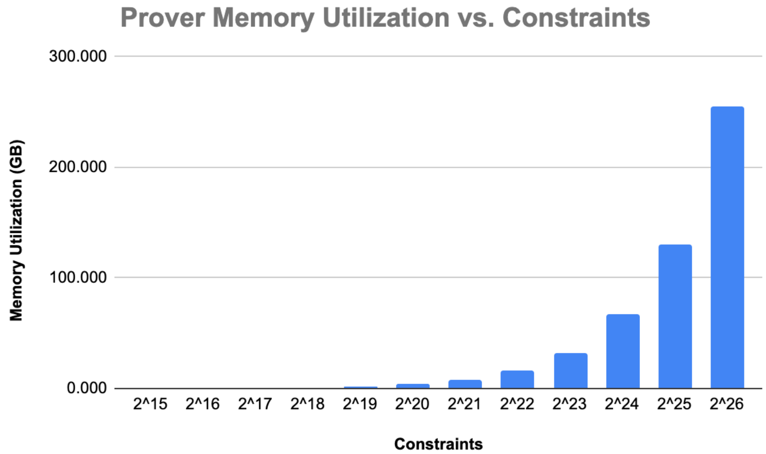





Construct Verifier Keys vs. Constraints
Execution Time (Regular)
Execution Time (ASM)
Execution Time (ASM + O3)
250
200
150
100
50
0
Execution Time (s)
2^15
2^16
2^17
2^18
2^19
2^20
2^21
2^22
2^23
2^24
2^25
2^26
Constraints


Proof Generation vs. Constraints
Execution Time (Regular)
Execution Time (ASM)
Execution Time (ASM + O3)
500
400
300
200
100
0
Execution Time (s)
2^15
2^16
2^17
2^18
2^19
2^20
2^21
2^22
2^23
2^24
2^25
2^26
Constraints


Proof Verification vs. Constraints
Execution Time (Regular)
Execution Time (ASM)
Execution Time (ASM + O3)
0.015
0.010
0.005
0.000
Execution Time (s)
2^15
2^16
2^17
2^18
2^19
2^20
2^21
2^22
2^23
2^24
2^25
2^26
Constraints



4 Future Research

Our results were obtained using one machine running on a single 128-core CPU. The next steps are executing these workloads on a single Nvidia GPU using Cuda.

5 References

[1] PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive Arguments of Knowledge: https://eprint.iacr.org/2019/953

[2] Barretenberg: https://github.com/AztecProtocol/barretenberg

[3] Ignition-Verification: https://github.com/AztecProtocol/ignition-verification

[4] Aztec's ZK-ZK-Rollup, Looking Behind the Cryptocurtain: https://medium.com/aztec-protocol/aztecs-zk-zk-rollup-looking-behind-the-cryptocurtain-2b8af1fca619