Benchmarking Groth16 Proving System

Lehigh University

Tal Derei and Ben Aulenbach

October 2022

1 Abstract

Zero-knowledge proofs are powerful tools that provide verification of a certain event or fact, without revealing information about the event or fact. This can be useful in financial applications, where borrowers can be accepted for a loan without revealing their exact income. It can also be useful in online voting, where a user can submit a proof of their vote without revealing who or what they voted for. These are just a couple examples of how zero-knowledge proofs can be used, with numerous other possibilities available. While they are useful in keeping sensitive information private, they are computationally heavy to generate. These proofs can be accelerated using modern parallel and distributed architectures like GPUs over thousands of cores. We summarize our preliminary benchmarks and areas to investigate further.

2 Codebase

The Mina Protocol, a blockchain network using zkSNARKs for verifiable data compression, previously held a competition to speed up **libsnark**, the industry standard C++ library for Groth16 [1]. The benchmark GPU prover implementation [2] is **2x** faster than the vanilla CPU prover. Aleo, a privacy-preserving blockchain environment for running private applications and smart contracts, later held a competition called **ZPrize** for speeding up Multi-Scalar Multiplication (MSM) on GPUs and FPGAs [3]**,** which we describe in section 4.3**.** Our benchmarks were collected against these repositories.

3 Cloud Computing Environment

We measure the relative speed up in performance across the following workstation, bare-metal machines instantiated on **Oracle Cloud**:

- ➔ **Phase 1:** 28-core Intel(R) Xeon(R) Gold 5120 CPU @ 2.20GHz, NVIDIA P100 (Pascal Architecture) with 12 GB HBM2, 192 GB DDR4 DRAM, 128 GB SSD.
- ➔ **Phase 2:** 64-core Intel(R) Xeon(R) Platinum 8358 CPU @ 2.60GHz, NVIDIA A10 (Ampere Architecture) with 24 GB GDDR6, 1024 GB DDR4 DRAM, 2 TB SSD.
- ➔ **Phase 3:** NVIDIA A40 (Ampere Architecture) with 48 GB GDDR6.

4 Experimental Results

The benchmark tooling includes the Oracle Cloud Infrastructure (OCI) monitoring service, alongside real-time process monitoring applications like htop and nvtop. All performance metrics are calculated based on 99th percentile statistics. The maximum constraint size for a program was 2^{25} constraints.

Our benchmarks fall into three distinct phases: Phase 1 (codenamed **Fracture**), Phase 2 (codenamed **Merge**), Phase 3 (codenamed **Star**). **Fracture** describes the process of running preliminary benchmarks on a sub-optimal computing setup, and the complications encountered during the benchmarking process. **Merge** represents the official benchmarks on a more advanced architecture set after resolving the issues in Fracture. **Star** represents benchmarks for Aleo's MSM algorithm, the current gold standard for this type of computation. Each phase is associated with a different OCI environment, and incorporates the following workloads: **1. Parameter Generation, 2. Preprocessing Generation**, **3. CPU Proof Generation, 4. GPU Proof Generation**.

Groth16 is a non-universal SNARK and requires a circuit-specific trusted setup. **Parameter Generation**, performed entirely on the CPU, describes the trusted-setup phase that produces the proving and verification keys, which are necessary public parameters. In

addition to generating the parameters, execution on the GPU requires an additional preprocessing step that precomputes multiples of the base points. This is the **Preprocessing Generation**. The precomputation can be reused over multiple MSMs (described below) with different parameters, saving overall work. More precomputation requires additional space, but accelerates each subsequent MSM. **Proof Generation** describes generating zero-knowledge proofs on CPUs and GPUs, involving multi-scalar multiplication (MSM) over elliptic curves and fast fourier transformations (FFTs) over large fields. MSM requires multiplications over large vectors and dominates 80-85% of the proof-generation time, while FFTs require complex polynomial calculations accounting for the other 10-15%. The relevant parameters for these workloads are described in the table below.

Parameter Generation	**Preprocessing Generation**	**CPU Proof Generation**	**GPU Proof Generation**
Execution Time	Execution Time	Execution Time	Execution Time
Memory Utilization	Memory Utilization	Memory Utilization	Memory Utilization
Input File Size	Input File Size	MSM Execution Time	MSM Execution Time
Parameter File Size	Preprocessing File Size	FFT Execution Time	FFT Execution Time
		CPU Utilization	GPU Utilization
			GPU VRAM Utilization

Table 1: Benchmarking Parameters

4.1 Phase 1: Fracture

Benchmarking the workloads described in **Table 1** initially presented a number of challenges with respect to our computing environment. Mina's Groth16 prover requires substantial disk and memory space, and our reference machine only included a 128 GB SSD and 192 GB DDR4 DRAM. These computing resources were evidently not large enough.

Parameter Generation	Preprocessing Generation	CPU Proof Generation	GPU Proof Generation
2^{25}	2^{23}	2^{25}	2^{23}
	2^{24}		2^{24}
	2^{25}		2^{25}

Table 2: Constraint Limits

Table 2 highlights the workloads that couldn't be performed past certain constraint sizes. Generating the parameters with greater than 2^{24} constraints exceeded main system memory, and the process was killed due to insufficient memory to perform the operation. Consequently, proof generation on the CPU failed for programs larger than 2^{24} constraints. Preprocessing the parameters (as required by the GPU prover) with greater than 2^{22} constraints generated a file that exceeded the system's boot volume, and the process was killed due to insufficient disk space. The GPU-based prover was unable to read in all the preprocessed curve points and aborted. As a result, proof generation on the GPU failed for programs larger than 2^{22} constraints.

4.2 Phase 2: Merge

All benchmarks were run using the MNT4-753 pairing-friendly elliptic curve over a finite field [4]. It is worth noting the MSM was executed on a single GPU, while the non-blocking FFTs are performed on the CPU in parallel. The maximum constraints size for a program was 2^{25} constraints.

4.2.1 Parameter and Preprocessing Generation

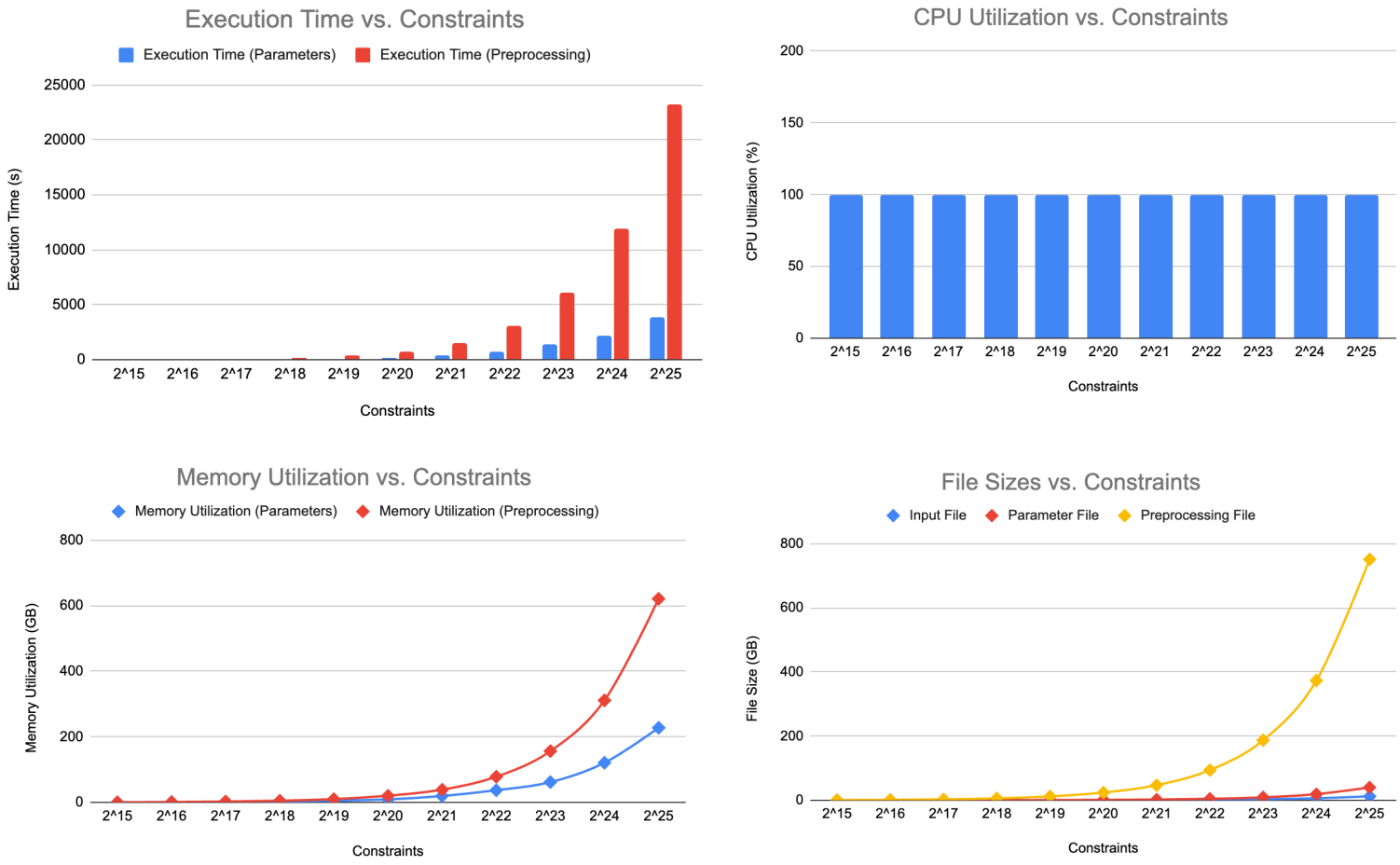


Figure 1: The reference machine is a 64-core Intel(R) Xeon(R) Platinum 8358 CPU @ 2.60GHz, NVIDIA A10 (Ampere) with 24 GB GDDR6, 1024 GB DDR4 DRAM, 2 TB SSD.

Figure 1 exemplifies that execution time and memory utilization grow exponentially with respect to the number of constraints in the program. This results in public parameters **3x** larger than the program's input. Preprocessing the parameters produces a file that grows super-exponentially with respect to the size of both the parameters and inputs. These parameters are ultimately injected into VRAM and system memory for the prover to access. The disk / memory size and execution time tradeoffs are steep as shown below.

4.2.2 Proof Generation

We measure the sustained CPU and memory utilizations for these workloads over the entire program runtime.

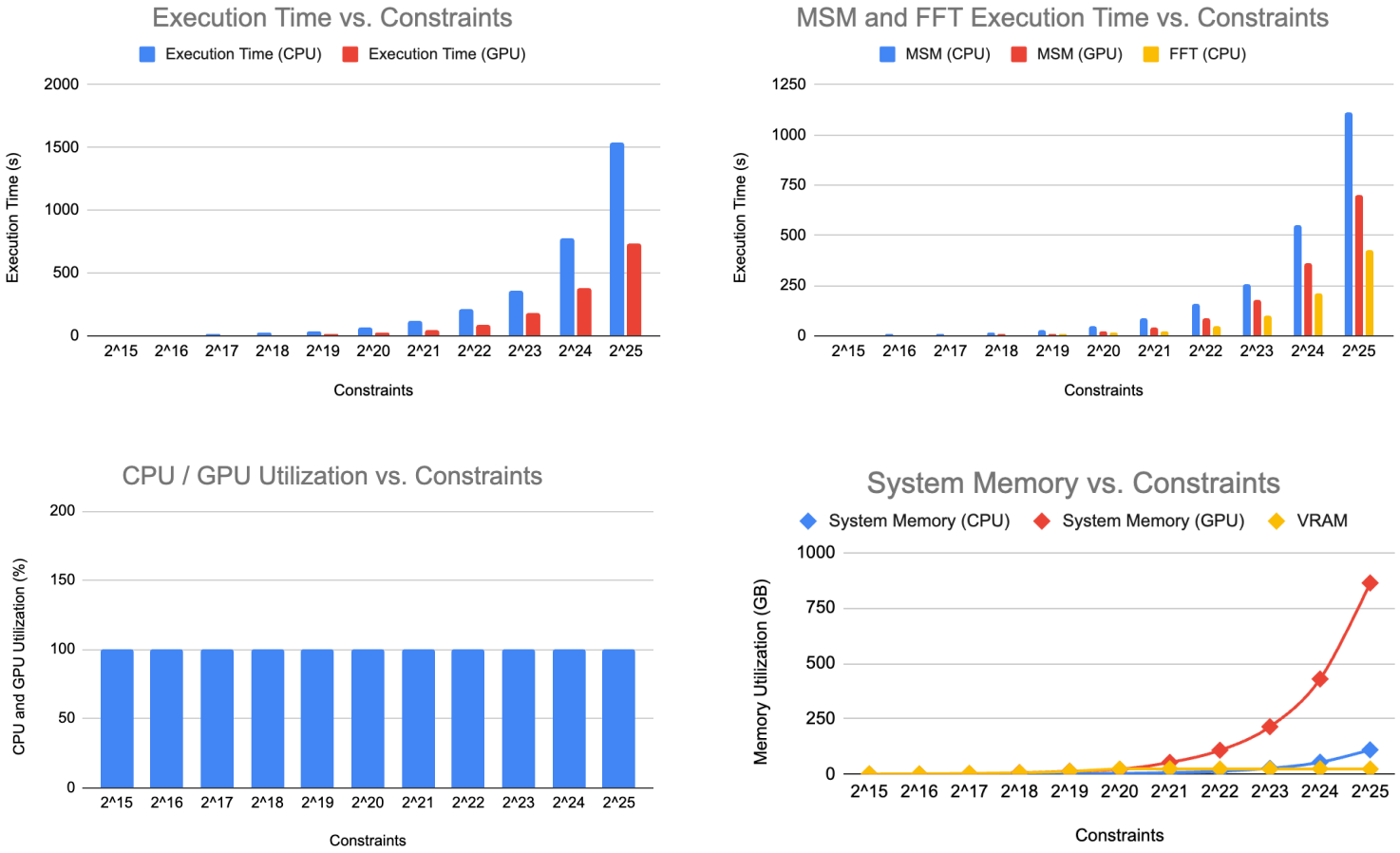


Figure 2: The reference machine is a 64-core Intel(R) Xeon(R) Platinum 8358 CPU @ 2.60GHz, NVIDIA A10 (Ampere) with 24 GB GDDR6, 1024 GB DDR4 DRAM, 2 TB SSD.

The results suggest the GPU-based prover executes roughly **2x** faster than the CPU-based prover. Additionally, execution time and memory utilization grow exponentially with respect to the number of constraints in the program. For programs larger than 2^{19} constraints, GPU VRAM is maxed out, which induces a spike in the system's main memory. Since the preprocessed parameter file is loaded into main memory, the parameters are then paged between host and device memory.

The MSMs and FFTs execute in parallel making it challenging to decouple their CPU and memory utilizations. On the CPU, the MSM dominates 85% of the prover runtime, and FFTs dominate the other 10-15%. On the GPU, the MSM and FFT runtimes are closer together since they execute in parallel. It is worth noting witness generation is not included in these benchmarks.

The results ultimately indicate that prover tradeoffs depend on the computing platform.. CPU-based provers use a small parameter file and less system memory, but execute slower. GPU-based provers use a larger preprocessed parameter file requiring more system memory and VRAM, but execute proportionally faster.

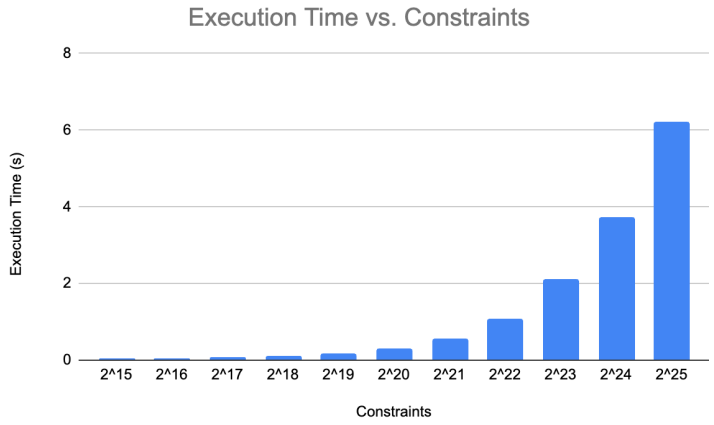
4.3 Phase 3: Star

Aleo's ZPrize competition for accelerating MSM operations on GPUs and FPGAs [5] will significantly advance the field. The multi-exponentiation algorithm represents a complex variation of Pippenger's algorithm. There is also no preprocessing step, which massively decreases the parameter file sizes loaded into VRAM and main memory.

The benchmarking harness can be configured to run multiple MSM batches of maximum size 2^{26}, all using the same points, but with different scalars. The results indicate MSM execution time is **~250x** faster than Mina's Groth16 prover for 2^{25} constraints on an A10 GPU. Aleo's codebase also uses **~2.5x** less memory for 2^{25} constraints on an A10 with

24 GB GDDR6 VRAM. The more performant A40 GPU computes a proof about **~1.5** faster than the A10, with similar memory profiles.

4.3.1 Parameter Generation



4.3.2 Proof Generation

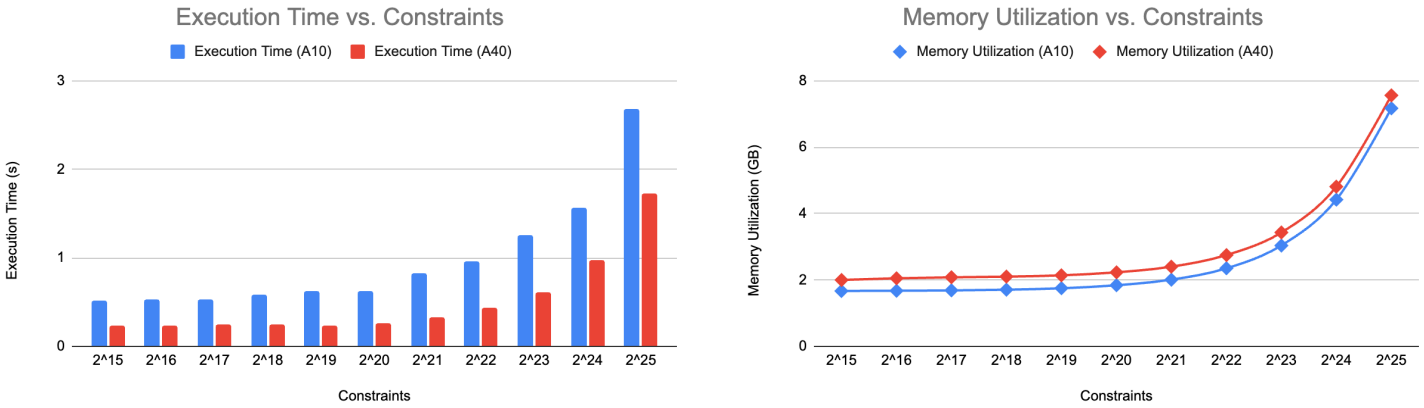


Figure 3: NVIDIA A40 (Ampere) with 48 GB GDDR6.

5 Future Research

Our results were obtained using one machine running on a single GPU. Having access to parallelism over multiple machines and GPUs will theoretically provide significant performance improvements.

Additionally, it is worth considering how the size of an elliptic curve affects prover runtime and determining the optimal balance between security and performance. The elliptic curve cannot be small enough to allow successful brute force attacks, but also cannot be too large, and unnecessarily slow performance.

With regards to the multi-exponentiation, it remains to be seen how this algorithm can be applied to PLONK for accelerating the execution time and reducing memory requirements.

6 References

[1] C++ CPU Groth16 Prover:
https://github.com/MinaProtocol/snark-challenge-prover-reference

[2] Cuda GPU Groth16 Prover:
https://github.com/MinaProtocol/gpu-groth16-prover-3x

[3] ZPrize's MSM Implementation:
https://github.com/z-prize/test-msm-gpu

[4] Elliptic Curves over Finite Fields:
https://github.com/TalDerei/Masters-Research/blob/main/Elliptic_Curves.pdf

[5] Aleo's ZPrize Competition:
https://www.zprize.io/prizes/accelerating-msm-operations-on-gpu-fpga