taiko

Standardizing zkEVM acceleration API

Mamy Ratsimbazafy

ZK Engineering @ Taiko

mratsim

m_ratsim

Agenda

- The state of the PSE (Ethereum Foundation) zkEVM ecosystem
- The end goal
- The journey
- What’s next?
- Collaborate

The state of PSE (Ethereum Foundation) zkEVM ecosystem

The users

- Co-processors:
  - Axiom
  - EZKL (zkML)

- Compilers, Languages & High-Level Frameworks
  - Lurk
  - Powdr

- Rollups:
  - Scroll
  - Taiko

The open-source HW accel landscape

- On Halo2-KZG (Privacy Scaling Exploration)
  - https://github.com/junyu0312/halo2 (March 2022)
  - https://github.com/privacy-scaling-explorations/halo2/pull/79 (June 2022)
  - https://github.com/superscalar-io/halo2_device_sample (October 2023)
- On Halo2-IPA (Zcash)
  - https://github.com/DelphinusLab/halo2-gpu-specific (October 2023)
- On Arkworks
  - Icicle (Ingonyama)
  - Spparks (Supranational)

The software landscape

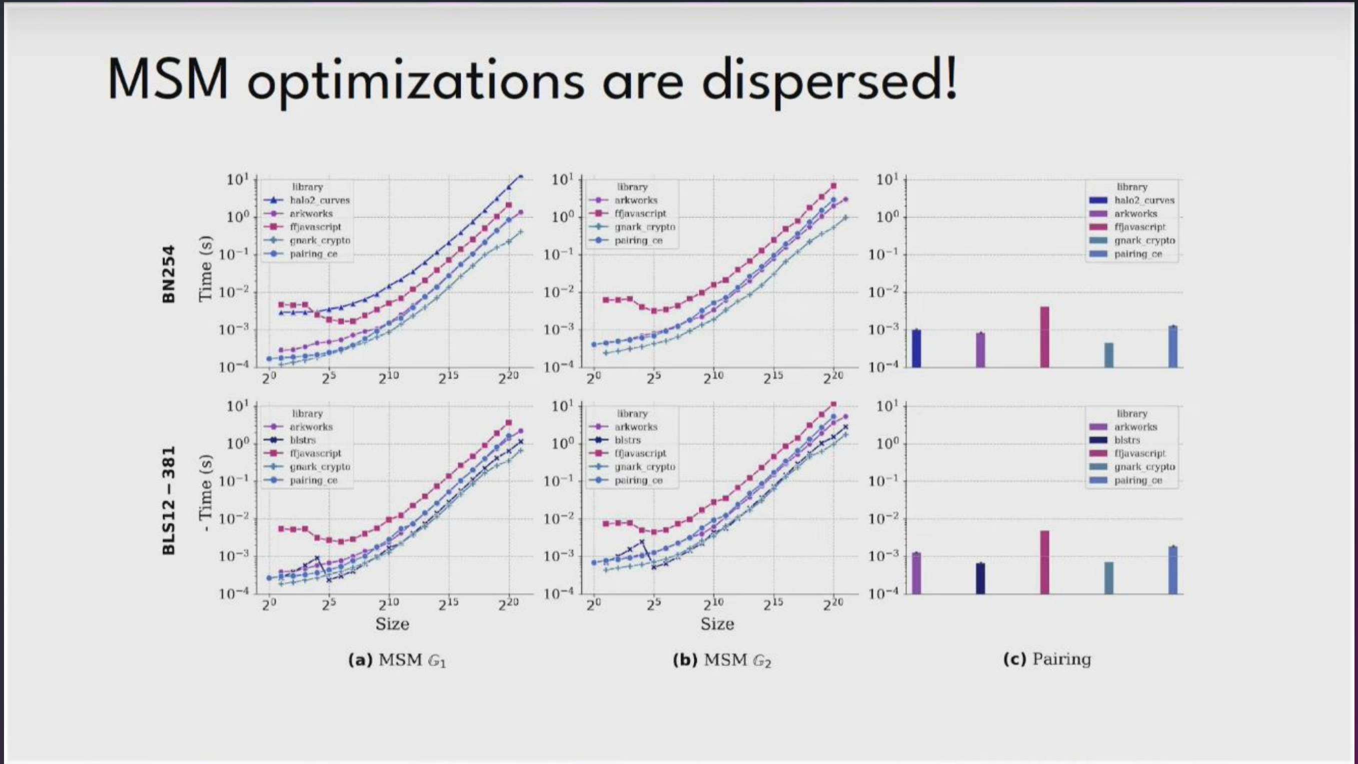
- Towards state-of-the-art MSM / halo2#187

- https://zka.lc/
  - Benchmarking SNARKS, zkSummit 10
    https://www.youtube.com/watch?v=fTGPUb07ebE
  - https://eprint.iacr.org/2023/1503
  - Gnark is the fastest MSM provider benchmarked

- Current speed, 2^22 / 4M points, i9-11980HK, 8 cores:
  PR halo2curves#86
  - Halo2-KZG (PSE): 3.5s
  - Gnark (Consensys): 1.2s
  - Constantine (Taiko): 1.2s

The software landscape

Ernstberger 2023 (zkSummit 10, Benchmarking SNARKS)



MSM optimizations are dispersed!

(a) MSM $\mathbb{G}_1$

(b) MSM $\mathbb{G}_2$

(c) Pairing

Software still has potential

- GPU renting costs
  - Competition with AI startups

- GPU embargos

- Parallelization on many-threaded CPUs
  - AMD EPYC 9654 96C/192T on 2 sockets hence 384 threads
  - Intel Xeon Platinum 8490H 60C/120T on 8 sockets hence 960 threads

- MSM parallelization level for the bucket method / Pippenger
  - MSM-level parallelism (partition points)
  - Window-level parallelism (partition scalar bits)
  - Bucket-level parallelism (need no collision when accumulating)

Software future developments

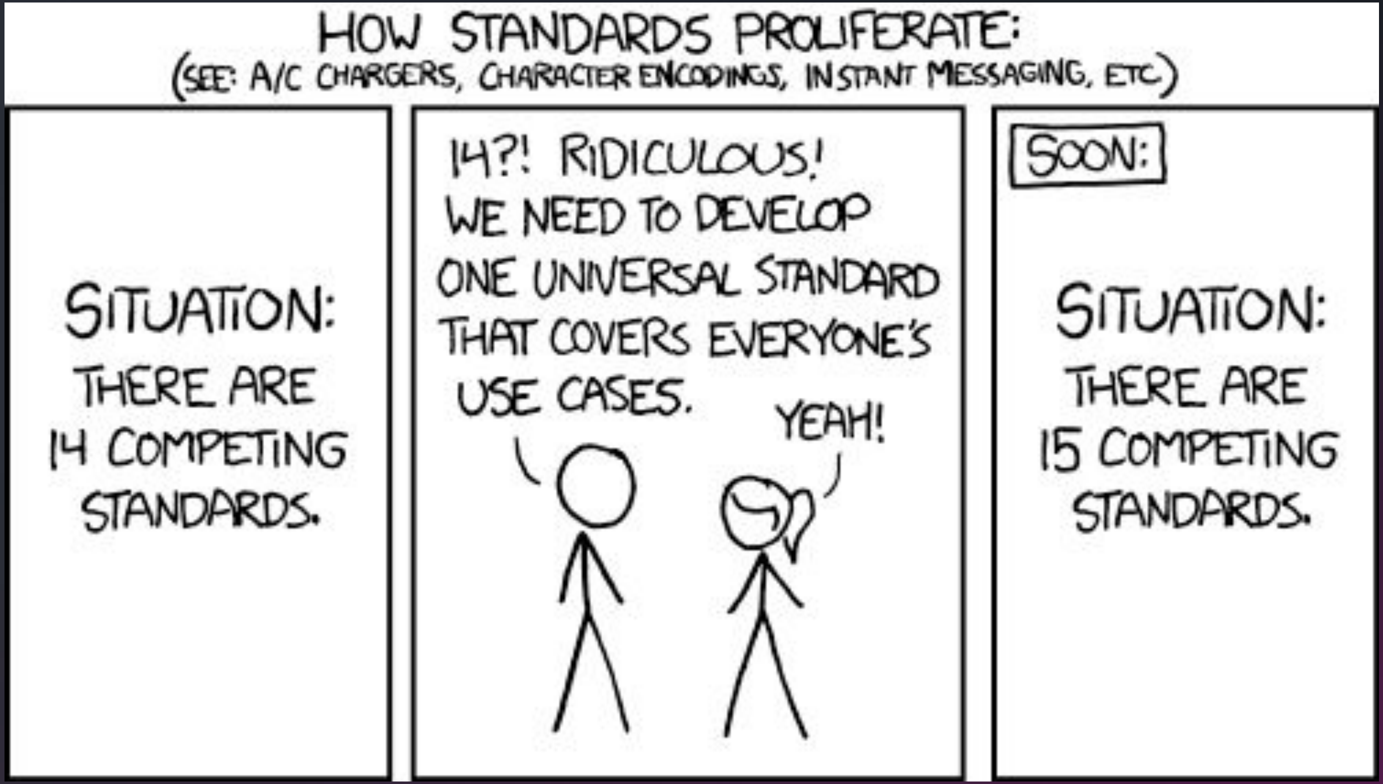
- Specialized MSM, example halo2#202 for bit-level keccak 40% speedup
- Prover-level parallelism, example EIP-4844 batch KZG commitments

$$e(\ \Sigma [r_i][\text{proof}_i]_1, [\tau]_2)\ .$$
$$e(\ \Sigma[r_i]([\text{commitment}_i]_1 - [\text{eval\_at\_challenge}_i]_1) + \Sigma[r_i][z_i][\text{proof}_i]_1, [-1]_2) = 1$$

https://github.com/mratsim/constantine/blob/5f7ba18f/constantine/commitments/kzg_polynomial_commitments_parallel.nim#L115

The end goal

The end goal


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

The end goal

- Disassociating proof systems from the computing backends
  - Provers are always evolving
  - The primitives they use MSMs, FFTs are not

- HW accel providers can serve many provers with a single codebase and small adapters instead of forking the complete codebase.
  - Less maintenance
  - Less bug surface
  - Focus on their specialty instead of having a proof-system person

- Proving libraries can focus on high-level proof-system, lookups optimizations

- Differential fuzzing and benchmarking

- End applications can pick the proving backend
  instead of being locked into the proving library default implementation

The journey

Standardizing the ABI

- Acceleration Abstraction Layer [Halo2#216]
- Proposes:
  - An encoding of field Fr elements
    (64-bit word-endianess, limb-endianess, saturated limbs, Montgomery representation)
  - An encoding of elliptic curve G1 elements
    (homogeneous projective coordinates, default in Halo2-KZG / Ingonyama vs jacobian coordinates in Halo2-IPA / Arkworks / Spark ...)
  - An async API to issue multiple MSMs or FFTs on the accelerator in parallel
  - A context parameter to abstract all accelerator configuration from the prover library

Standardizing the ABI

- Inspired by accelerators for image processing and machine learning
  - Nvidia Cuda and CuDNN
  - Intel oneDNN
  - OpenCV HAL (Hardware Abstraction Layer)
- Windows Kernel API

- An engine context
- An optional operation descriptor
  - We probably let the accel library handle memory allocation unlike what drivers or kernel do.

Changes in Halo2

- `best_fft` appears twice
- `best_multiexp` appears 9 times

- All callers will need to pass a new `engine` context parameter
- End application will need to init the `engine` context with
  - Number of cores used (including reading env variables like TAIKO_NUM_THREADS)
  - GPU devices used

What’s next

Future improvements

- Async API for issuing multiple MSMs / FFTs in parallel (like EIP-4844 batch verification in Constantine)
- Caching abstraction via operation-specific descriptors / context
- Commitment-level Acceleration Layer
  - Instead of Arithmetic-level Acceleration Layer

Collaborate

Collaborate

- https://github.com/privacy-scaling-explorations/halo2/issues/216
- https://github.com/superscalar-io/halo2_device_sample

- Working group on Telegram or Discord?
  - Hardware providers
  - Software libraries (Halo2-IPA, Halo2-KZG, Artworks, ...)
  - End-users (co-processors, languages & frameworks, rollups)

The End

taiko

Standardizing zkEVM acceleration API

Mamy Ratsimbazafy

ZK Engineering @ Taiko

mratsim m_ratsim