Enabling AI ASICs for Zero Knowledge Proof

Jianming Tong*
jianming.tong@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Jingtian Dang*
dangjingtian@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Simon Langowski
slangowski@mit.edu
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA

Tianhao Huang
tianhaoh@mit.edu
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA

Asra Ali
asraa@google.com
Google
Austin, Texas, USA

Jeremy Kun
jkun@google.com
Google
Portland, Oregon, USA

Jevin Jiang
jevinjiang@google.com
Google
Sunnyvale, California, USA

Srinivas Devadas
devadas@mit.edu
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA

Tushar Krishna
tushar@ece.gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Abstract

Zero-knowledge proof (ZKP) provers remain costly because multi-scalar multiplication (MSM) and number-theoretic transforms (NTTs) dominate runtime as they need significant computation. AI ASICs such as TPUs provide massive matrix throughput and SotA energy efficiency. We present *MORPH*, the first framework that reformulates ZKP kernels to match AI-ASIC execution. We introduce *Big-T* complexity, a hardware-aware complexity model that exposes heterogeneous bottlenecks and layout-transformation costs ignored by Big-*O*. Guided by this analysis, (1) at arithmetic level, MORPH develops an MXU-centric extended-RNS lazy reduction that converts high-precision modular arithmetic into dense low-precision GEMMs, eliminating all carry chains, and (2) at dataflow level, MORPH constructs a unified-sharding layout-stationary TPU Pippenger MSM and optimized 3/5-step NTT that avoid on-TPU shuffles to minimize costly memory reorganization. Implemented in JAX, MORPH enables TPUv6e8 to achieve up-to 10× higher throughput on NTT and comparable throughput on MSM than GZKP. Our code: https://github.com/EfficientPPML/MORPH.

1 Introduction

Zero-knowledge proofs (ZKPs) have evolved from a theoretical construct into a core building block for practical verifiable computation and scalable blockchains. They allow services to outsource computation while keeping data private and verification cheap, but the **prover remains prohibitively expensive**, e.g., generating a proof for ImageNet ViT requires nearly one hour [42]. Across widely deployed protocols [16, 17], prover runtime is dominated by **multi-scalar multiplication (MSM)** and the **number-theoretic transform (NTT)**, which commonly account for ≈70% and 20–30% of total latency, respectively [19, 41].

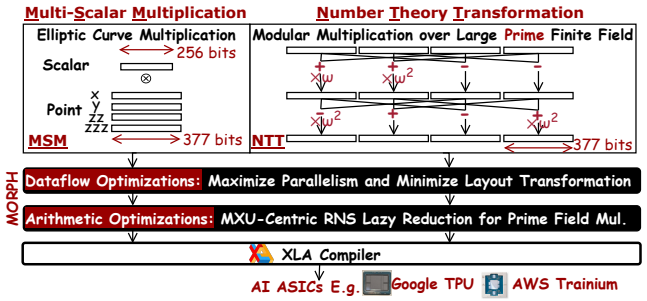


Figure 1: MORPH's deployment flow with dataflow and arithmetic optimizations to accelerate ZKP on TPU.

Both MSM and NTT expose abundant parallelism at practical ZKP sizes, making parallel hardware promising for acceleration. Among these platforms, AI ASICs (Google TPU [20], AWS Trainium [4] etc.) offer extreme compute density and energy efficiency, significantly outperforming general-purpose accelerators at scale [37]. Architecturally, a TPU contains giant heterogeneous cores, including a matrix multiplication engine (MXU) and a vectorized processing engine (VPU). Each MXU/VPU contains orders of magnitude (1024/16×) more multiply-accumulates (MACs) than one corresponding core in a GPU [8]. All MACs in a MXU/VPU execute instruction in the lock-step (SIMD) to substantially reduce per-operation control overhead and improve energy efficiency. This paper studies how to **systematically repurpose AI ASICs, particularly Google's TPUs [21], for ZKP kernels (MSM/NTT)** and achieve higher energy efficiency and SoTA throughput at scale.

Existing ZKP acceleration spans GPUs [13, 19, 28, 29, 43], FPGAs [7, 26, 30–34] and ASICs [35, 38, 41]. While these systems hand-craft MSM/NTT algorithms for their respective platforms, they provide *no principled way to quantify how far an MSM/NTT algorithm is from the platform's performance limits with detailed reasoning*. Some works provide Big-O complexity [23], yet we find a surprising phenomenon: the fastest GPU/TPU algorithms intentionally use the algorithm with higher Big-O to expose massive parallelism and thus achieve higher throughput [37]. *Big-O alone is insufficient for modern heterogeneous parallel hardware.*

Furthermore, Big-O cannot capture layout transformation cost, which becomes the dominant bottleneck on AI ASICs. When we

*Both authors contributed equally to this research.

DAC '26, Long Beach, CA, USA

ACM ISBN 979-8-4007-2254-7/2026/07
https://doi.org/10.1145/3770743.3804402

port the SotA MSM and NTT algorithms for GPU/FPGA/ZKP-ASIC to TPUs with similar overall throughput, they slow down by 30× in our evaluation. This is because a TPU ***only* operates on coarser-grained contiguous data (a 4KB vector register, VReg)**, and fine-grained data gathering in SotA MSM/NTT algorithms requires expensive shuffles and transposes to move data into contiguous VRegs. These layout-induced stalls are invisible under the traditional Big-O model. To address this gap, we introduce Big-T notation, a platform-aware complexity model that measures the sequential bottleneck across heterogeneous pipelined compute units and the latency of layout transformation, providing an asymptotic lower bound for parallel execution on AI ASICs. Big-T exposes two structural inefficiencies in SotA MSM/NTT algorithms on TPUs.

Arithmetic-level Challenge: ZKP requires modular arithmetic over large prime fields (e.g., 256/377/753 bits), but hardware natively supports much lower precision (e.g., 8/32-bit). Because prime fields lack a natural RNS factorization, the SotA algorithm [6, 28] performs high-precision arithmetic via digit decomposition and a chain of carry propagation, achieving <1% compute utilization on TPUs.

Dataflow-level Challenge: The SotA dataflows incur prohibitive communication and storage overheads. In presorted Pippenger MSM [19, 28], sorting is distributed across many GPU CUDA cores. Directly porting this strategy to TPU introduces prohibitive inter-TPU communication. Layout-invariant 3-step NTT [37] requires parameters that scale linearly with problem size, resulting in out-of-memory (OOM) failures at large scale.

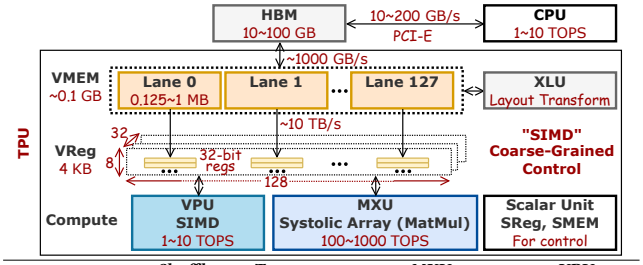
In resolving these inefficiencies, we propose **MORPH**[1], the *first* framework enabling TPU as ZKP accelerator with arithmetic and dataflow level optimizations, as shown in Fig. 1.

Arithmetic-level Solution: Since a prime field cannot be represented in a Residue Number System (RNS), MORPH adopts [18, 25] to encapsulate modular multiplication in a prime field in a *larger non-prime finite field* that has a RNS representation. This enables high-precision multiplication to be expressed as independent 32-bit vector multiplications with no carry propagation. Then, MORPH reforms the modular reduction (RNS reconstruction → modular reduction → RNS decomposition) as low-precision matrix multiplication and parallel vectorized operations to be accelerated by TPU's MXU and VPU, achieving up-to 90× speedup.

Dataflow-level Solution: To tackle communication and layout reorganization, MORPH proposed **LS-PPG** to schedule communication-free workload dimensions across devices to avoid inter-TPU communication, ensures layout invariant within individual TPU. To mitigate parameters storage overflow, MORPH's **5-step NTT** recursively partitions workloads to reduce parameter size while preserving the same computational complexity.

Together, MORPH enables TPUv6e-8 to achieve up-to 10× NTT throughput and 1.2× MSM throughput than GZKP [28] (NVIDIA V100). **MORPH makes TPU the SotA throughput machine for high-precision NTT**, a promising platform for ZKP acceleration. Our contributions are:

- **BIG-T Complexity:** A hardware-aware complexity metric that captures heterogeneous compute spans and layout transformation, enabling principled analysis of MSM/NTT designs on AI ASICs and guiding the construction of layout-stationary pippenger for MSM and 5-step NTT, giving 5.8× and 1.6× speedup.
- **MXU-centric RNS Lazy Reduction:** A matrix-oriented modular reduction for big integer multiplication in prime fields, which replaces sequential carry-propagation arithmetic with MXU-accelerated RNS-lazy reduction, achieving up-to 90× speedup over the SotA radix decomposed Montgomery reduction.
- **LS-PPG and 5-step Layout Invariant NTT:** Both avoid inter-TPU communication and minimize intra-TPU layout reorganization, delivering up-to 5.8× speedup over the SoTA MSM algorithm[19] on TPU, and reduce parameters storage at scale.



	Shuffle	Transpose	MXU	VPU
Max Parallelism	$PAR_S = 4096$	$PAR_T = 4096$	$PAR_{MXU} = 131072$	$PAR_{VPU} = 2048$
Datatype	u32	u32	bf16/i8	u32
Requirement	(8, 128)	(8, 128)	(8×128×128 MatMul)	2048

Figure 2: TPU programming model. All compute units operate on VRegs with (8×128) 32-bit registers each. Performance hinges on data being laid out such that tiles needed for computation can be loaded directly from VRegs. Values from [21].

2 Background and Related Work

2.1 Tensor Processing Unit (TPU)

The TPU [21] is Google's AI ASIC designed to accelerate machine learning training and inference with high energy efficiency. All on-chip computation is organized around the **Vector Register (VReg)** abstraction. Each VReg consists of **8×128** of **32**-bit values executed in lock-step SIMD. All compute units on a TPU, including the VPU, MXU, and XLU, consume and produce data strictly at VReg granularity, as shown in Fig. 2. This uniform but coarse layout enables high efficiency but restricts flexibility: workloads whose shapes do not align with the fixed (8, 128) structure incur layout transformation because only full VRegs can be processed each cycle.

• **Vector Processing Unit (VPU).** The VPU performs 32-bit SIMD arithmetic on VRegs. E.g., TPUv4 contains a dual-issue ALU to operate two VRegs concurrently, yielding a peak parallelism of $P_{VPU} = 2048$ 32-bit operations per cycle. Vectors with length not a multiple of (8, 128) cannot fully utilize VPU's available parallelism.

• **Matrix Multiplication Unit (MXU).** The MXU in TPUv4 is a large systolic array (128×128) for int8 and bf16 matrix multiplication. It preloads weight tiles from VRegs and streams activation tiles from VRegs. When normalized to the same datatype, the MXU achieves roughly **16× higher peak throughput** than the VPU. Full efficiency requires workloads to conform to the MXU's minimum effective tile of **8×128×128**. Larger gives more weights reuse while smaller or non-aligned matrix shapes leave MXU under-utilized.

• **Layout Transformation Unit (XLU).** XLU performs shuffles, transposes, broadcasts, and reductions of VRegs. XLU is efficient for transformations at VRegs granularity, but costly for element-wise

[1] MORPH: Matrix-Oriented Reformulation of ZKP for Parallel Heterogeneous AI ASICs

permutations. Shuffling N individual elements may cost N **cycles**.
• **HBM** and the **host CPU** serve as the off-chip data sources, but their bandwidth is typically **an order of magnitude lower** than on-chip VReg bandwidth. Consequently, high-performance TPU kernels favor **on-chip data reuse and layout transformation**.

2.2 Performance Metric and Analysis

TPU's heterogeneous compute units, each with different parallelism and tiling constraints, make classical models such as Big O and the work–span model inadequate. These models measure total computational work or critical-path latency under the assumption of a homogeneous sequential machine like a CPU, and therefore cannot capture VReg granularity, systolic-array tiling, or the cost of on-chip layout transformations. Further, Roofline analysis[39] and trace-based profiling quantify how far execution is from hardware limits, but they still do not explain why performance is low.

2.3 Zero-knowledge Proof (ZKP) Acceleration

2.3.1 *Background on Kernel and Arithmetic in ZKP.*

Multi-Scalar Multiplication. MSM [17] computes $\sum_{n=0}^{N-1} S_n \circledast P_n$ where each scalar S_n is S^{BL} bits and each point P_n lies on an elliptic curve that supports point addition (PADD, $\oplus$). A scalar–point multiplication ($\circledast$) is realized as repeated PADDs for S_n times.

Number Theory Transformation. NTT [17] converts a polynomial of degree N into its frequency-domain representation by evaluating it at N distinct roots of unity in a finite field. Given an input vector $\mathbf{x}$ of degree N, NTT computes $\mathbf{X}_k = \sum_{n=0}^{N-1} x_n \times \omega_N^{kn} \bmod M$, $k \in [0, N)$, where ω_N is a primitive N-th root of unity.

Large prime field. Modern ZKP systems [17] rely on arithmetic over a finite field $\mathbb{F}_M$ with a **large prime** modulus M (256/377/753 bits). In both elliptic-curve MSM and polynomial-commitment NTT, all additions, multiplications are executed modulo large prime M.

2.3.2 *Related Work: Arithmetic Implementations.*

ZKPs operate over large prime fields $\mathbb{F}_M$ with $\log_2(M) \geq 256$, exceeding native 32-bit word size on GPUs and TPUs. SotA GPU implementations [19, 28] therefore represent each field element as a vector of D 32-bit "digits". A single field multiplication requires two stages: (1) *High-precision multiplication*, which performs $O(D^2)$ 32-bit multiplications followed by a $O(D)$ sequential carry propagation of length D; and (2) *Montgomery reduction*, which reduces the $2D$-digit intermediate product back to D digits, incurring another $O(D^2)$ 32-bit multiplications. We use this SotA radix Montgomery reduction as our **arithmetic baseline**, termed as ***radix Mont***.

RNS represents an integer x by residues ($x \bmod q_i$) over coprime moduli $\{q_i\}$, where modulus $Q = \prod_i q_i$. RNS could lower $O(D^2)$ down to $O(D)$ when performing modular multiplication with respect to Q. But the prime modulus M cannot be factored into multiple residues, making such reduction inapplicable to ZKP.

2.3.3 *Related Work: Dataflow Optimizations.*

Presorting Pippenger (PPG) - MSM In prior work of MSM acceleration on GPU [19, 29], FPGA [7, 26, 31–33] and ASICs [12, 27, 38, 40, 41], Presorting PPG is the SotA MSM algorithm. Its key idea is to *reduce the total number of point additions* by first grouping points whose scalars share the same value, summing those points once, and then multiplying the accumulated result by that scalar value. To increase the likelihood of such sharing ("collisions"), each scalar is split into $K = \lceil S^{BL}/c \rceil$ windows of c bits. We adopt presorting PPG as our **dataflow baseline**, termed as **"Presort-PPG"**.

Butterfly NTT and layout invariant 3-step NTT. SotA NTT acceleration on CPUs [11, 24], GPUs [13, 28, 43], FPGAs [34] and ASICs [30] implements recursive Cooley-Tukey NTT with minimal computation complexity $O(NlogN)$, commonly called as "butterfly NTT" in Fig. 5a. However, each stage performs fine-grained shuffles that are smaller than vector-register width, leading to costly layout transformations, making TPUs inefficient. Recent TPU work [37] proposes a *layout-invariant 3-step NTT* achieving $O(N^{3/2})$ arithmetic with zero layout transformation, breaking the NTT throughput record of the butterfly NTT on GPUs [15, 22] on lower-precision $M < 32$ bits and lower degree ($N \leq 2^{17}$). It reforms an N-input NTT into an (R, C) matrix ($N = R \times C$) such that an NTT is reformed into matrix and vectorized multiplication for TPU acceleration, as shown in Fig. 5b. **We adopt 3-step NTT as dataflow baseline.**

3 Method

This section first defines the Big T notation, then uses it to analyze arithmetic and dataflow inefficiencies of the SotA MSM/NTT algorithm on TPUs, finally showing how MORPH resolves them.

3.1 Big-T Complexity and TPU's Parallelism

Today's parallel computing devices, including GPUs and TPUs, often consist a heterogeneous set of pipelined compute units $\mathcal{U} = \{U_1, U_2, \ldots, U_K\}$, where each U_k denotes a class of units. Let P_k be the number of parallel instances of unit type U_k. We decompose the total work of size N into per-unit contributions $W_k(N), k \in [1, \ldots, K]$, where $W_k(N)$ counts the operations mapped to unit U_k.

All these on-chip compute units are deeply pipelined, the execution time is bounded by the slowest (bottleneck) pipeline stage. We define the *BIG-T complexity* (termed as Big-T notation) of an algorithm on a parallel computing device as

$$T(N) = O\Big(\max\Big\{\max_k \frac{W_k}{P_k},\ Mem\Big\}\Big)$$

if there exists a constant $c > 0$ and N_0 such that for all $N \geq N_0$, $T(N) \leq c \cdot \max\Big\{\max_k \frac{W_k}{P_k},\ Mem\Big\}$. Here, the term "$\max_k \frac{W_k}{P_k}$" captures the bottleneck among heterogeneous pipelined compute units. "Mem" captures the overall latency of off-chip data access. **Together, they provide an asymptotic lower latency bound of a proposed algorithm on parallel computing devices.** Under sufficiently large workload, the ideal parallelism of compute units in a TPU used for Big-T analysis is listed in Fig. 2.

3.2 Arithmetic Optimizations

Table 1: Big-T Complexity of Arithmetic (Bottleneck in Red)

Kernel	VPU Span	MXU Span	XLU Span	Memory Span
Radix Mont.	$\frac{D^2}{PAR_{\text{VPU}}}$	$\frac{D^2}{PAR_{\text{MXU}}}$	$\frac{D^2 \log D}{PAR_{\text{S}}}$	$\frac{D}{BW_{\text{HBM}}}$
MXU RNS Lazy	$\frac{4D}{PAR_{\text{VPU}}}$	$\frac{D^2}{PAR_{\text{MXU}}}$	~0	$\frac{2D}{BW_{\text{HBM}}}$

3.2.1 *Challenge: Radix Montgomery Reduction is Dominated by Memory Reorganization.*

For a value with D 32-bit digits, radix-2^{32} Montgomery multiplication requires (1) a $O(D^2)$ big-integer multiplication and (2) a $O(D^2)$ Montgomery reduction, each containing a sequential D-step carry-propagation chain. Carry propagation demands fine-grained shuffling and index permutation on a TPU,

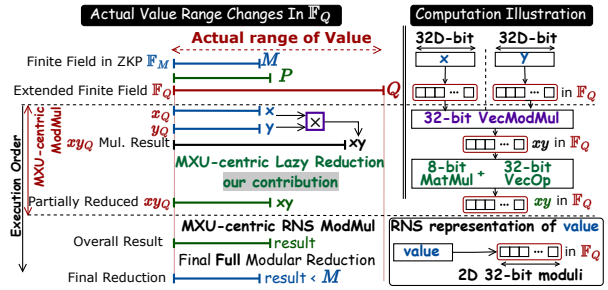


Figure 3: Illustration of actual value change and computational flow of *MXU-centric RNS Lazy Modular Multiplication.*

which appears as the XLU span bottleneck in Tab. 1. Thus, the limiting factor is not compute throughput, but the repeated memory reorganization inherent in Radix Montgomery Reduction.

3.2.2 Solution: Reducing $O(D^2) \rightarrow O(2D)$ for Big-Integer Multiplication via RNS. To eliminate quadratic-cost digit multiplications, we encode each high-precision prime-field element in an extended non-prime field $\mathbb{F}_Q$ following [18]. We select $Q > M^2$ to guarantee that for any $x, y \in \mathbb{F}_Q$, the product of their RNS vectors never overflows Q, avoiding a true reduction by Q. Hence, the multiplication is fully decomposed into independent 32-bit limb multiplications. *This transformation converts the compute pattern from quadratic all-to-all digit multiplications into linear digit-independent multiplications.*

*3.2.3 Solution: Reducing $O(D^2)$ Reduction via **MXU-Centric RNS Lazy Reduction**.* Although multiplication is linearized in $\mathbb{F}_Q$, modular reduction by M must still be performed over the prime field $\mathbb{F}_M$. A naïve approach would require RNS reconstruction from $\mathbb{F}_Q$ $\rightarrow$ prime-field reduction (mod M) $\rightarrow$ RNS decomposition into $\mathbb{F}_Q$.

We reduce this overhead by applying Basis Aligned Transformation [37] to Simon's reduction [25], collapsing this multi-stage pipeline into one 8-bit matrix multiplication (Lines 15 and 18 in Alg.1) plus four 32-bit vector operations (Lines 16,17,19,20 in Alg.1). This shifts the dominant $O(D^2)$ term to a low-precision matrix multiplication, with its cost amortized by MXU, yielding $O(D)$ latency overhead. It also removes all carry-propagation from the critical path, making it throughput-bound rather than shuffle-bound.

3.2.4 Walk-Through Example. Given $x, y \in \mathbb{F}_M$, we convert them into extended-field RNS form x_Q, y_Q, each containing $2D$ 32-bit digits. The modular multiplication proceeds as:

- RNS-level 32-bit VecMul with Montgomery reduction, $O(2D)$.
- MXU-centric RNS Lazy Reduction (Lines 14–20 of Alg.1, $O(D^2)$).

All sequential carry propagation are removed. Computation becomes dominated by vectorized operations, as highlighted in Tab. 1.

In summary, **MORPH encapsulates computation over $\mathbb{F}_M$ inside a larger field $\mathbb{F}_Q$, paying** (1) a 2× memory footprint for the extended RNS representation, and (2) additional RNS reconstruction/decomposition. These costs are amortized among sea-of-MACs in MXU, such that the bottleneck shifts from sequential shuffle-bound carry propagation to VPU-bounded vectorized operations, enabling better throughput/latency than Radix Mont.

3.3 Dataflow Optimizations

3.3.1 Multi-Scalar Multiplication (MSM).

Challenge. porting SotA MSM to TPU leads to three inefficiencies: (1) *inter-TPU communication*, (2) *intra-TPU layout reorganization*, and (3) *memory overhead* from loading duplicated points of different windows from off-chip memory.

MORPH introduces *Layout-Stationary Pippenger* (**LS-PPG**), which enforces a common sharding and layout across presorting and Bucket Accumulation (BA), the latency-dominant phase of MSM. LS-PPG shards reduction-free dimensions across tensor cores, allowing each core to process an independent partition. This makes presorting a direct producer of BA-ready points, eliminating cross-TPU communication, intra-TPU layout reorganization. By fusing presorting with BA, post-sorted points are consumed immediately upon generation rather than being written back to and later reloaded from off-chip HBM. Beyond BA, MORPH performs Bucket Reduction in a tree form to expose more parallelism and improve VPU compute utilization. Alg. 2 and Fig. 4 illustrate LS-PPG. Consequently, the ideal memory span is reduced from $\frac{KN}{BW_{HBM}}$ to a single pass over scalars and points, i.e., $\frac{2N}{BW_{HBM}}$ in Tab. 2.

Algorithm 1 MXU-Centric RNS Lazy Reduction **(MXU RNS Lazy)**

ByteDecompose$(x) \rightarrow [x]_b, b \in [0, B)$

1: **for** $b = 0$ **to** $B - 1$:
2: $\quad x_b \leftarrow x\,[8b : 8(b+1)]$ ▷ Select b-th byte of x, full parallel
3: **Return** $[x]_b, 0 \le b < B$

ByteMerge$([x]_h, h \in [0, H)) \rightarrow x$

4: **for** $h = 0$ **to** $H - 1$:
5: $\quad x\,[8h : 8(h+1)] = [x]_h$ ▷ Parallel among H bytes
6: **Return** x

Precomputation(Q, P, w)

Require: Q; P; w: the Montgomery factor used in element wise reduction; $y = (2^w)^{-1}$, $z = 2^w$, y and z are co-prime.
7: Initialize I_i, $I_{i,b}$, $[E]_{i,b,j,h}$, $[f]_i$.
8: $\quad I_i = \left(\left(\left(\frac{Q}{q_i}\right)^{-1} \bmod p_i\right)\left(\frac{Q}{q_i} y\right)\right) \bmod Q$
9: $I_{i,b} = \left(\left(\left(\frac{Q}{q_i}\right)^{-1} \bmod p_i\right)\left(\frac{Q}{q_i} y\right)\right) << 8b \bmod Q$ ▷ BAT[37]
10: $[E]_{i,b,j,h} = \text{ByteDecompose}\left(\left(z(I_{i,b} \bmod M)\right) \bmod p_j\right)_h$
11: $[f]_i = \left\lceil \frac{I_i 2^u}{Q} \right\rceil$
12: $[g]_j = (-z(Q \bmod M)) \bmod p_j$
13: **Return** $[E]_{i,k}$, $[f]_i$, $[g]_j$

Main(x_Q, Q, P, w) ▷ MXU-centric RNS Lazy Reduction

Require: $Q = \prod_i q_i$; $P = \prod_j p_j, i \in [0, I), j \in [0, J)$; w; pick $u \ge \lceil log2(\sum_i q_i) \rceil$. $x_Q = \text{CRT}_Q(x)$ is RNS representation of x in $\mathbb{F}_Q$, containing I residues with B Bytes each. When being written as $\left[x_Q\right]_i$, each iteration specifies entire 32-bit residue value of x_Q. When being written as $\left[x_Q\right]_{i,b}$, $b \in [0, B)$, each iteration specify b-th byte of i-th residue. $[x_P]_{j,h}$ has J residues with H bytes each. Note: we refer to [25] for details of CRT_Q.
Ensure: $x_P = ((((x \times y) \bmod M) \bmod Q) \times z) \bmod P$ in $\mathbb{F}_P$
14: $[E]_{i,b,j,h}$, $[f]_i$, $[g]_j \leftarrow \text{Precomputation}(Q, P, w)$
15: $v \leftarrow \left[x_Q\right]_i \cdot [f]_i$ ▷ Dot Product, fused into L18 as one MatMul
16: $k \leftarrow \left\lfloor \frac{v}{2^u} \right\rfloor$ ▷ 32-bit Vectorized Shift
17: $\left[x_Q\right]_{i,b} = \text{ByteDecompose}(\left[x_Q\right]_i)$
18: $[x_P]_{j,h} = \sum \left[x_Q\right]_{i,b} \times [E]_{i,b,j,h}$ ▷ (Einsum) uint8 MatMul
19: $[x_P]_j = \text{ByteMerge}([x_P]_{j,h})$
20: **Return** $[x_P]_j + k \cdot [g]_j$ ▷ 32-bit ScalarVecMul + VecAdd

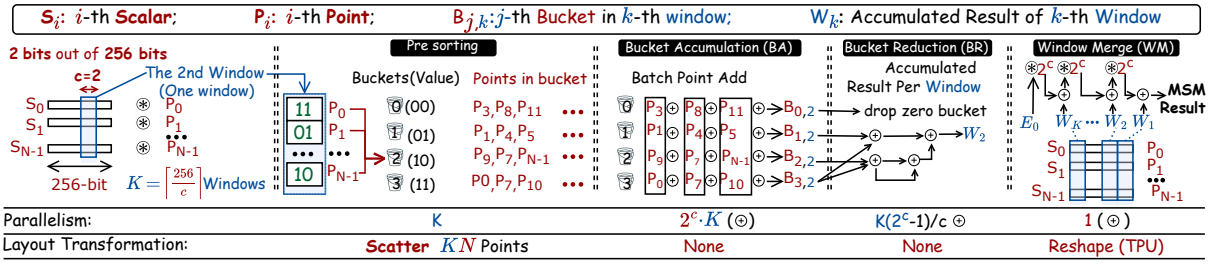
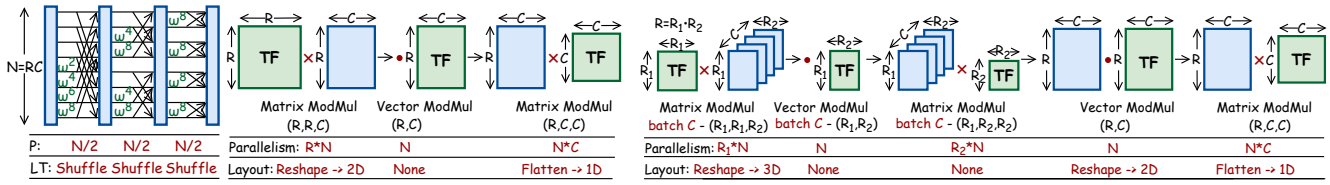


Figure 4: Illustration of LS-PPG's (Alg. 2). Takeaway: MORPH minimizes data re-sharding and layout reorganization.



(a) Butterfly NTT **(b) 3-step NTT** **(c) 5-step NTT (Proposed by This Work)**

Figure 5: Performance Analysis of Different NTT. ($N = R \times C$, $R = R_1 \times R_2$), TF refers to twiddle factors, which are generated offline. *Takeaway: MORPH further recursively reduces the row-wise NTT in (b) into an 3-step NTT to reduce overall computation.*

Algorithm 2 Layout Stationary Pippenger (**LS-PPG**)

Require: Initialize $MSM, S_n, P_n, W_k, B_{k,j}, W_k$ as 0, $j \in [0, 2^c - 1], k \in [0, K]$, N'(max point count per $B_{k,j}$), definition in Fig. 4.

`Bucketize`

sharding for $k \in K$ ▷Distributed across multi-TPUs
  Initialize $P'_k \leftarrow \mathcal{O}$ of shape $[2^c, N']$ ▷ 2^c buckets.
  $\Pi_k \leftarrow$ `argsort`(I_k) ▷I_k: k-th slice of all scalars S_n
  Gather P_n based on Π_k, and scatter them into $I_k[n]$-th bucket in k-th window (W_k).

`Bucket Accumulation (BA)`

Initialize bucket $B_{k,j}$ with zero points.
sharding for $k \in K$ ▷Distribute windows across multi-TPUs
  parallel for $j \in [1, 2^c - 1]$
    for $n' \in N'$
      $B_{k,j}$ += $P'_{k,n',j}$

`Bucket Reduction (BR)` ▷ Tree-based Accumulation

Initialize $W_{k,j} \leftarrow \mathcal{O}$ for all $k \in [0, K-1]$ and $j \in [0, 2^c - 1]$
for $s = c - 1$ down to 0 ▷level of tree
  Let $B^{(L)}_{k,b} \leftarrow B_{k,2b}$ and $B^{(R)}_{k,b} \leftarrow B_{k,2b+1}$ for $b \in [0, 2^s - 1]$
  Let $W^{(L)}_{k,b} \leftarrow W_{k,2b}$ and $W^{(R)}_{k,b} \leftarrow W_{k,2b+1}$ for $b \in [0, 2^s - 1]$
  sharding for $b = 0$ to $2^s - 1$
    parallel for $k = 0$ to $K - 1$
      $W_{k,b} \leftarrow W^{(L)}_{k,b} + W^{(R)}_{k,b} + B^{(R)}_{k,b}$
      $B_{k,b} \leftarrow 2 \cdot \left(B^{(L)}_{k,b} + B^{(R)}_{k,b}\right)$
$W_k \leftarrow W_{k,0}$ ▷ root of the tree

`Window Merging (WM)`

for k in **(K, 0, -1)**: ▷**Reverse** Window Index k
  $MSM \times= 2^c$
  MSM += W_k
Return MSM.

Note 1: Coordinate and bytes dimensions are hidden for simplicity.
Note 2: $\mathcal{O}$ is the zero point $(0, 1, 1, 0)$ in Twisted Edward curves.
Note 3: $B^{(L)}_{k,b}, B^{(R)}_{k,b}, W^{(L)}_{k,b}, W^{(R)}_{k,b}$ are introduced for tree reduction.
Note 4: N' is the maximal number of points among all buckets.

Table 2: Big-T Complexity of Dataflows (Bottleneck in Red)

MSM	VPU Span and MXU Span	XLU Span	Memory Span
Presort-PPG	$\frac{KN}{PAR^{BA}} + \frac{2K(2^c-1)}{PAR^{BR}} + \frac{(K-1)(1+C)+1}{PAR^{WM}}$	$\frac{2^c \cdot N \log N}{PAR_S}$	$\frac{K \cdot N}{BW_{HBM}}$
LS-PPG	$\frac{KN}{PAR^{BA}} + \frac{4K(2^c-1)}{PAR^{BR\_new}} + \frac{(K-1)(1+C)+1}{PAR^{WM}}$	$\frac{2^c \cdot N \log N}{PAR_S}$	$\frac{2N}{BW_{HBM}}$
NTT	**VPU Span**	**MXU Span**	**XLU Span**	**Memory Span**
Butterfly	$\frac{N \log N}{PAR_{VPU}}$	0	$\frac{N \log N}{PAR_S}$	$\frac{(N+N)}{BW_{HBM}}$
3-step NTT	$\frac{N}{PAR_{VPU}}$	$\frac{N(R+C)}{PAR_{MXU}}$	$\frac{2N}{PAR_T}$	$\frac{(2N+R^2+C^2)}{BW_{HBM}}$
5-step NTT	$\frac{2N}{PAR_{VPU}}$	$\frac{N(R_1+R_2+C)}{PAR_{MXU}}$	$\frac{3N}{PAR_T}$	$\frac{(2N+R_1^2+R_2^2+R+C^2)}{BW_{HBM}}$

Note: $PAR^{BA}/PAR^{BR}/PAR^{WM}$ refers to PAR_{VPU} or PAR_{MXU} in BA,BR,WM phase. $PAR^{BR\_new} = c$. $PAR^{BR} = 2$. VPU / MXU runs the same number of PADD.

3.3.2 *Number-Theoretic Transformation (NTT).*

Challenge of Butterfly NTT and 3-step NTT. Tab. 2 shows the Big-T complexity of NTT variants. Running Butterfly NTT on a TPU shows $\frac{PAR_{VPU}}{PAR_S} \approx O(10^3)$, making XLU the critical bottleneck slowing down butterfly NTT by $O(10^3)$. This is slow even though it offers minimal compute complexity. The 3-step NTT [37] removes shuffles by reforming a N-degree NTT into two (R, R, C) matrix multiplications and N-length vector operations, where $N = R \cdot C$. However, its Big-T is dominated by the MXU span $N(R+C)/PAR_{MXU}$, which grows with N by a factor of $(R+C)$. Even with balanced factors ($R = C = \sqrt{N}$), it incurs a memory span of at least $4N/BW_{HBM}$.

Solution: 5-step NTT to reduce MXU span while preserving MXU utilization. To further reduce the MXU and memory span, MORPH introduces a **5-step NTT** that replaces the row-wise R-degree NTT (step 1 in the 3-step NTT of Fig. 5b) with a 3-step NTT over (R_1, R_2, C), as shown in Fig. 5c, where $R = R_1 \cdot R_2$. This decomposes the large GEMM into multiple smaller GEMMs, reducing the effective MXU span by a factor of $N^{1/4}$ while being sufficiently large to fully utilize the MXU. The 5-step NTT is detailed in Eq. 1.

$$TF^R_{R\times R} @ \left[TF^{C\times R_2}_{R_2\times R_2} @ \left[\left(a_{R_1\times C\times R_2} @ TF^{C\cdot R_1}_{R_1\times R_1}\right) \cdot TF^{C}_{R_1\times R_2} \right] \cdot TF_{C\times C} \right] \quad (1)$$

Here, @ denotes matrix multiplication and $\cdot$ represents element-wise multiplication. $TF^k_{R\times C}$ is a matrix $[((\omega_n)^k)^{ij}], i \in [0, R), j \in [0, C)$. ω_n: primitive N-th root of unity. a is the input tensor.

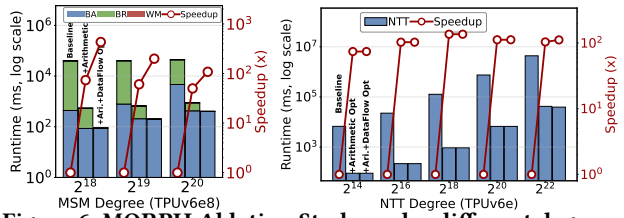


Figure 6: MORPH Ablation Study under different degrees.

4 Experiments

4.1 Experiment Setup

Workload: We take the most popular primitives from zk-SNARKs (degree usually ranges from $2^{14} \sim 2^{26}$) [3, 9, 14, 36]. We adopt the same evaluation setup as GZKP for MSM and NTT [1, 28], where we assume data comes in affine representation [2] and offline converted into Twisted Edwards form to minimize compute overhead.

TPU Baseline: We adopt SotA GPU algorithms as our baseline, including (1) radix Montgomery Reduction [19, 28], (2) Presorting Pippenger [19], and (3) 3-step NTT [37], detailed in §2.3.3.

MORPH TPU Setup: MORPH uses (1) MXU-centric RNS Lazy Reduction, (2) LS-PPG, and (3) 3-step and 5-step NTT. MORPH is implemented in JAX [10] and captures wall-clock latencies using the JAX Profiler (XProf) [5]. We use Google TPUv6e as platforms, and scale the number of chips to match V100's power consumption.

4.2 MORPH Evaluation

We conduct three-stage ablation for MSM and NTT (Baseline, +Arithmetic, +Ari. + Dataflow,) with latency and speedup plotted in Fig. 6.

4.2.1 Arithmetic Optimization Evaluation. Arithmetic optimizations yield **50** $\sim$ **90**× latency reduction for MSM, collapsing BR/WM to near-constant cost. These gains come from **MXU-centric RNS Lazy Reduction**, which removes sequential carry propagation and reformulating modular reduction of big integer arithmetic into low-precision dense matrix multiplication, allowing the MXU to amortize native $O(D^2)$ computations into $O(D)$ latency. Although this increases memory footprint by up to 2×, MSM remains compute-bound on VPU operations, so off-chip latency is effectively hidden. Overall, MXU-accelerated RNS-lazy arithmetic is the **primary contributor** to performance improvement for both MSM and NTT.

4.2.2 MSM – Dataflow Optimization Evaluation. MORPH enables up-to 3.1× speedup. Across all degrees, **BA (Bucketized included)** dominates the runtime, because of random scatter and gather. Dataflow optimizations reduce BR by $\sim$ 6× from parallelism increase.

4.2.3 NTT – Dataflow Optimization Evaluation. **3-step NTT is faster at lower degrees**, while **5-step becomes faster at higher degrees with up-to 1.07× speedup**. 5-step NTT forces the overall degree N to be decomposed into three factors. At small degree, these factors are typically $\leq 2^6 = 64$, leading to poor utilization of the $(8, 128)$ VReg shape and reducing utilization for the MXU and VPU. As the degree grows ($> 2^{22}$), these factors become large enough to fully utilize VRegs, and 5-step benefits from reduced storage overhead of twiddle parameters.

4.3 MORPH vs. SotAs

4.3.1 NTT. Tab. 3 highlights two core advantages of TPUs for large-precision NTTs. **First, TPUv6e8 achieves SoTA throughput in NTT**: Its leading energy efficiency transfers to 10× higher throughput than GZKP (V100) for 753-bit NTTs. **Second, TPUs scale more gracefully with precision.** Increasing the modulus from 256- to 753-bits raises GPU latency by 6× $\sim$ 7× due to long Montgomery carry chains, whereas TPU latency grows only 1.3× $\sim$ 3× because RNS arithmetic maps efficiently onto the MXU. TPUs also avoid butterfly shuffling entirely. Their cost is determined by the compute spans in Tab. 2, including MatMul $(N(R+C))/PAR_{\text{MXU}}$, XLU twiddle steps ($2N \sim 3N$), and $(2N + R^2 + C^2)/BW_{\text{HBM}}$ memory span, all of which are small for lower-degree NTTs.

Table 3: Latency Comparison (MORPH -tpuv6e8 v.s. GZKP-V100). Unit: μs/ms for NTT/MSM. Improvement in red.

Workload	Scale	753-bit		256-bit	
		GZKP	MORPH	GZKP	MORPH
NTT (μs)	2^{14}	150	15.1	50	11.6
	2^{16}	490	52.8	90	24.8
	2^{18}	910	337	280	109
	2^{20}	7,460	2,195	1,070	716
	2^{22}	33,670	13,107	4,960	4,694
	2^{24}	141,400	OOM	20,990	21,382
	Throughput ↑		**2.6–10×**		**0.98–4.3×**
MSM (ms)	$N = 2^{22}$	2.66	2.24	0.24	1.61
	$N = 2^{24}$	11.30	8.91	1.10	6.42
	$N = 2^{26}$	40.70	35.64	4.00	25.64
	Throughput ↑		**1.14–1.2×**		**0.15–0.16**

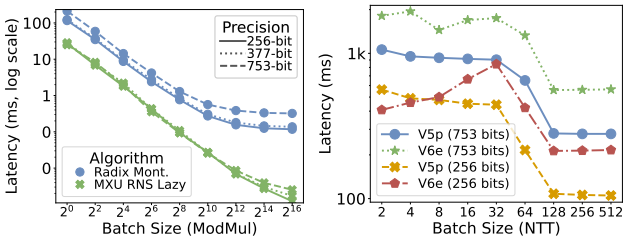


Figure 7: ModMul and NTT under different batch sizes.

4.3.2 MSM. Tab. 3 reports estimated latency relative to GZKP. MORPH 's LS-PPG allows TPUv6e8 to achieve competitive throughput for 753-bit MSM. The latency is dominated by Bucketize, whose per-point scatter exhibits poor memory-bandwidth utilization. This overhead becomes more severe with finer-grained point scatter, leading to lower performance than GZKP at 256-bit.

4.4 Ablation Study - Batch Size

MXU-centric RNS-Lazy sustains **4**$\sim$**157**× lower latency than Radix Montgomery across all batch sizes and precisions, and the performance gap widens as precision increases (256⟶377⟶753 bits) and as batch size increases, it further amplifies MXU utilization (Fig. 7).

Increasing batch size from 1 to 128 gives **3.8/3.2**× and **5.4/3.9**× speedup on NTT (753 and 256 bit) for TPUv5p/v6e (Fig. 7). **NTT favors** ≥ 128 **batches** to fully utilize VRegs. Latency plateaus beyond 128 batch size as MXU/VPU achieves its peak compute utilization.

5 Conclusion

This paper presents MORPH, the first framework to accelerate ZKP kernels on AI ASICs, deployed on Google TPU. Using a Big-T formulation, we identify the two core bottlenecks and show how to overcome them: bridging high-precision modular arithmetic (>256-bit) to low-precision matrix/vector engines via an **extended non-prime RNS field**, and eliminating or hiding layout-reorganization and inter-device communication overhead through **dataflow with layout-invariant sharding over non-reduction dimensions**. MORPH makes TPU the SotA commodity platform for high-precision NTT throughput and position AI ASICs as a practical foundation for full ZKP protocol acceleration.

5.1 Acknowledgments

This work was supported in part by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. We thank reviewers for their feedbacks.

References

[1] [n. d.]. Benchmark harness for FPGA MSM implementations in the ZPRIZE competition. https://github.com/z-prize/prize-gpu-fpga-msm/tree/main/harness. Accessed: April 1, 2025.

[2] [n. d.]. XYZZ coordinates for short Weierstrass curves. https://www.hyperelliptic.org/EFD/g1p/auto-shortw-xyzz.html. Accessed: April 9, 2025.

[3] [n. d.]. ZK Rollup Architecture. Online. https://zksync.io/faq/tech.html#zk-rollup-architecture Accessed: April 2025.

[4] 2025. Amazon Trainium. https://aws.amazon.com/ai/machine-learning/trainium/. Accessed: [Insert Date Accessed, e.g., April 9, 2025].

[5] 2025. Perfetto Trace Viewer. https://perfetto.dev/. Accessed: [Insert Date Accessed, e.g., April 9, 2025].

[6] 2025. yrrid GPU MSM Library. https://github.com/yrrid/combined-msm-gpu. Accessed: [Insert Date Accessed, e.g., April 9, 2025].

[7] Kaveh Aasaraai, Don Beaver, Emanuele Cesena, Rahul Maganti, Nicolas Stalder, and Javier Varela. 2022. *CycloneMSM: FPGA Acceleration of Multi-Scalar Multiplication.* Technical Report. IACR. https://eprint.iacr.org/2022/1396.pdf.

[8] Jacob Austin, Sholto Douglas, Roy Frostig, Anselm Levskaya, Charlie Chen, Sharad Vikram, Federico Lebron, Peter Choy, Vinay Ramasesh, Albert Webson, and Reiner Pope. 2025. How to Scale Your Model. Online. (2025). Retrieved from https://jax-ml.github.io/scaling-book/.

[9] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. *Zerocash: Decentralized Anonymous Payments from Bitcoin.* Technical Report. Zerocash Project. http://zerocash-project.org/media/pdf/zerocash-extended-20140518.pdf

[10] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. *JAX: composable transformations of Python+NumPy programs.* http://github.com/jax-ml/jax

[11] Jeff Buss et al. 2021. Intel HEXL: High-Performance Homomorphic Encryption Primitives. In *Proceedings of the Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC).*

[12] Alhad Daftardar, Brandon Reagen, and Siddharth Garg. 2024. SZKP: A Scalable Accelerator Architecture for Zero-Knowledge Proofs. In *Proceedings of the 2024 International Conference on Parallel Architectures and Compilation Techniques* (Long Beach, CA, USA) *(PACT '24)*. Association for Computing Machinery, New York, NY, USA, 271–283. doi:10.1145/3656019.3676898

[13] Wei Dai and Berk Sunar. 2015. cuHE: A Homomorphic Encryption Accelerator Library. *IACR Cryptology ePrint Archive* 2015 (2015), 1043.

[14] Dark Forest Team. [n. d.]. Announcing Dark Forest. Blog post. https://blog.zkga.me/announcing-darkforest Accessed: April 2025.

[15] Shengyu Fan, Zhiwei Wang, Weizhi Xu, Rui Hou, Dan Meng, and Mingzhe Zhang. 2022. TensorFHE: Achieving Practical Computation on Encrypted Data Using GPGPU. arXiv:2212.14191 [cs.AR] https://arxiv.org/abs/2212.14191

[16] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. 2019. Plonk: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. *IACR Cryptol. ePrint Arch.* 2019 (2019), 953. https://eprint.iacr.org/2019/953.

[17] Jens Groth. 2016. On the Size of Pairing-based Non-interactive Arguments. In *Advances in Cryptology - EUROCRYPT 2016 (Lecture Notes in Computer Science, Vol. 9666)*. Springer, 305–326. doi:10.1007/978-3-662-49896-5_11

[18] David Jacquemin, Ahmet Can Mert, and Sujoy Sinha Roy. 2022. Exploring RNS for Isogeny-based Cryptography. Cryptology ePrint Archive, Paper 2022/1289.

[19] Zhuoran Ji, Zhiyuan Zhang, Jiming Xu, and Lei Ju. 2024. Accelerating Multi-Scalar Multiplication for Efficient Zero Knowledge Proofs with Multi-GPU Systems. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 57–70. doi:10.1145/3620666.3651364

[20] Norman P. Jouppi et al. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. arXiv:1704.04760 [cs.AR] https://arxiv.org/abs/1704.04760

[21] Norman P. Jouppi et al. 2023. TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. arXiv:2304.01433 [cs.AR] https://arxiv.org/abs/2304.01433

[22] Jongmin Kim, Wonseok Choi, and Jung Ho Ahn. 2024. Cheddar: A swift fully homomorphic encryption library for cuda gpus. *arXiv preprint arXiv:2407.13055* (2024).

[23] Jongmin Kim, Sangpyo Kim, Jaewan Choi, Jaiyoung Park, Donghwan Kim, and Jung Ho Ahn. 2023. SHARP: A Short-Word Hierarchical Accelerator for Robust and Practical Fully Homomorphic Encryption. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23)*. Association for Computing Machinery, New York, NY, USA, Article 18, 15 pages. doi:10.1145/3579371.3589053

[24] Kim Laine, Rachel Player, and Hao Chen. 2018. Microsoft SEAL: A Homomorphic Encryption Library. In *Proceedings of the IEEE Symposium on Security and Privacy Workshops (SPW)*. IEEE, 123–126.

[25] Simon Langowski and Srinivas Devadas. 2025. Efficient Modular Multiplication Using Vector Instructions on Commodity Hardware. Cryptology ePrint Archive, Paper 2025/1068. https://eprint.iacr.org/2025/1068

[26] Changxu Liu, Hao Zhou, Patrick Dai, Li Shang, and Fan Yang. 2023. PriorMSM: An Efficient Acceleration Architecture for Multi-Scalar Multiplication. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. doi:10.1145/3678006 https://dl.acm.org/doi/10.1145/3678006.

[27] Changxu Liu, Hao Zhou, Lan Yang, Jiamin Xu, Patrick Dai, and Fan Yang. 2024. Gypsophila: A Scalable and Bandwidth-Optimized Multi-Scalar Multiplication Architecture. In *Proceedings of the 61st ACM/IEEE Design Automation Conference* (San Francisco, CA, USA) *(DAC '24)*. Association for Computing Machinery, New York, NY, USA, Article 94, 6 pages. doi:10.1145/3649329.3658259

[28] Tianyu Ma, Zhen Zhang, Yuhao Zhang, and G. Edward Suh. 2023. gZKP: GPU-Accelerated Zero-Knowledge Proof Generation. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE.

[29] Weiliang Ma, Qian Xiong, Xuanhua Shi, Xiaosong Ma, Hai Jin, Haozhao Kuang, Mingyu Gao, Ye Zhang, Haichen Shen, and Weifang Hu. 2023. GZKP: A GPU Accelerated Zero-Knowledge Proof System. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS 2023)*. Association for Computing Machinery, Vancouver, BC, Canada, 340–353. doi:10.1145/3575693.3575711

[30] Mustafa Mert, Daniel Gilad, and Christopher W. Fletcher. 2022. F1: A Fast and Programmable Accelerator for Fully Homomorphic Encryption. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1168–1183.

[31] Ayumi Ohno, Kotaro Shimamura, and Shinya Takamaeda-Yamazaki. 2025. Accelerating Elliptic Curve Point Additions on Versal AI Engine for Multi-scalar Multiplication. arXiv:2502.11660 [cs.AR] https://arxiv.org/abs/2502.11660

[32] Xander Pottier, Thomas de Ruijter, Jonas Bertels, Wouter Legiest, Michiel Van Beirendonck, and Ingrid Verbauwhede. 2025. OPTIMSM: FPGA hardware accelerator for Zero-Knowledge MSM. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2025, 2 (2025), 489–510.

[33] Andy Ray, Benjamin Devlin, Fu Yong Quah, and Rahul Yesantharao. 2023. Hardcaml MSM: A High-Performance Split CPU-FPGA Multi-Scalar Multiplication Engine. In *Proceedings of the ACM Symposium on Field-Programmable Gate Arrays*. doi:10.1145/3626202.3637577 https://dl.acm.org/doi/10.1145/3626202.3637577.

[34] Brandon Reagen, Woojoo Choi, David Brooks, Gu-Yeon Wei, and Hsien-Hsin S. Lee. 2021. HEAX: An Architecture for Computing on Encrypted Data. In *Proceedings of the ACM/IEEE International Symposium on Computer Architecture (ISCA)*. IEEE, 1113–1126.

[35] Nikola Samardzic, Simon Langowski, Srinivas Devadas, and Daniel Sanchez. 2024. Accelerating Zero-Knowledge Proofs Through Hardware-Algorithm Co-Design. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 366–379. doi:10.1109/MICRO61859.2024.00035

[36] Roman Storm, Alexey Pertsev, and Roman Semenov. 2020. Tornado Cash Privacy Solution. https://tornado.cash/Tornado.cash_whitepaper_v1.4.pdf White Paper.

[37] Jianming Tong, Tianhao Huang, Jingtian Dang, Leo de Castro, Anirudh Itagi, Anupam Golder, Asra Ali, Jeremy Kun, Jevin Jiang, Arvind, G. Edward Suh, and Tushar Krishna. 2026. Leveraging ASIC AI Chips for Homomorphic Encryption. In *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1–18. doi:10.1109/HPCA68181.2026.11408507

[38] Cheng Wang and Mingyu Gao. 2025. UniZK: Accelerating Zero-Knowledge Proof with Unified Hardware and Flexible Kernel Mapping. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 1101–1117. doi:10.1145/3669940.3707228

[39] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (April 2009), 65–76. doi:10.1145/1498765.1498785

[40] Zhengbang Yang, Lutan Zhao, Peinan Li, Han Liu, Kai Li, Boyan Zhao, Dan Meng, and Rui Hou. 2025. LegoZK: A Dynamically Reconfigurable Accelerator for Zero-Knowledge Proof. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 113–126. doi:10.1109/HPCA61900.2025.00020

[41] Ye Zhang, Shuo Wang, Xian Zhang, et al. 2021. PipeZK: Accelerating Zero-Knowledge Proof with a Pipelined Architecture. In *Proceedings of the 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*.

[42] Yancheng Zhang, Mengxin Zheng, Xun Chen, Jingtong Hu, Weidong Shi, Lei Ju, Yan Solihin, and Qian Lou. 2025. zkVC: Fast Zero-Knowledge Proof for Private and Verifiable Computing. *arXiv preprint arXiv:2504.12217* (2025).

[43] Zhenyu Zhao, Xin Chen, Kai Chen, and Dingyi Zou. 2021. High-Performance Polynomial Arithmetic for Lattice-Based Cryptography on GPUs. In *Proceedings of the International Conference on High Performance Computing (HiPC)*. IEEE.