SumcheckPIM: An Efficient HBM-Based PIM Architecture for Linear Complexity Zero Knowledge Proofs

Sunchae Kim
Division of Engineering Science
University of Toronto
Toronto, Canada
sunchae.kim@mail.utoronto.ca

Taewoon Kang
Computer Science and Engineering
Korea University
Seoul, Republic of Korea
taewoon_kang@korea.ac.kr

Sangwon Shin
Computer Science and Engineering
Korea University
Seoul, Republic of Korea
husask11@korea.ac.kr

Taeweon Suh
Computer Science and Engineering
Korea University
Seoul, Republic of Korea
suhtw@korea.ac.kr

Yibin Yang
Electrical and Computer Engineering
University of Toronto
Toronto, Canada
yibiny@ece.utoronto.ca

Gunjae Koo
Computer Science and Engineering
Korea University
Seoul, Republic of Korea
gunjaekoo@korea.ac.kr

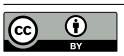
Abstract

Zero-knowledge proofs (ZKPs) are emerging as a core technology for privacy-preserving computation. Despite steady progress in protocol and algorithm design, generating these proofs remains computationally intensive, driving growing interest in hardware acceleration for kernels such as number-theoretic transform (NTT) and multi-scalar multiplication (MSM). Among them, the sumcheck protocol offers a compelling alternative with $O(n)$ prover complexity compared to $O(n \log n)$ for NTT-based approaches, yet our analysis reveals its execution is fundamentally memory-bound, with severely underutilized compute resources. This characteristic demands a memory-centric acceleration strategy, in contrast to compute-centric approaches of prior work.

To this end, we propose SumcheckPIM, an HBM-based processing-in-memory architecture optimized for the sumcheck protocol. SumcheckPIM introduces 256-bit modular arithmetic units at the bank level to support real-world field sizes, a Fiat-Shamir unit on the logic die for global accumulation and challenge generation, and an inter-bank processing engine enabling protocol completion without host interaction. We also propose a DRAM-aware folding scheme that maximizes row buffer locality through row packing and bank ping-ponging. Our design achieves this while maintaining JEDEC compliance with minimal modifications to existing commercial HBM-PIM architectures. We demonstrate that SumcheckPIM delivers both high performance and efficiency, achieving 5.3×–13.3× and 334.2×–361.1× speedup over GPU and CPU baselines, respectively.

Keywords

zero-knowledge proof, sumcheck protocol, processing-in-memory, near data processing



ICS '26, Belfast, United Kingdom

ACM ISBN 979-8-4007-2522-7/26/07
https://doi.org/10.1145/3797905.3807843

ACM Reference Format:
Sunchae Kim, Taewoon Kang, Sangwon Shin, Taeweon Suh, Yibin Yang, and Gunjae Koo. 2026. SumcheckPIM: An Efficient HBM-Based PIM Architecture for Linear Complexity Zero Knowledge Proofs. In *2026 International Conference on Supercomputing (ICS '26), July 06–09, 2026, Belfast, United Kingdom.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3797905.3807843

1 Introduction

Zero-knowledge proofs (ZKPs) [36] enable a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that a statement is true without revealing anything beyond its validity. ZKPs provide two key capabilities: (i) *privacy*, by proving claims about sensitive data without disclosing it, and (ii) *verifiability*, by enabling others to check correctness much more efficiently than re-executing the underlying computation.

ZKPs can be built as interactive protocols that require multiple rounds of communication between $\mathcal{P}$ and $\mathcal{V}$, or non-interactive protocols where $\mathcal{P}$ generates a proof that can be verified offline. Non-interactive ZKPs, particularly zk-SNARKs (Zero-Knowledge Succinct Non-interactive Argument of Knowledge), have been widely deployed due to their succinct proof size and efficient verification. In this work, we focus on zk-SNARK-style systems, for which proof generation is typically the dominant cost.

In a zk-SNARK system (depicted in Figure 1), $\mathcal{P}$ takes a public statement and a private witness as input, runs a proving algorithm to generate a succinct proof, and sends it to $\mathcal{V}$. $\mathcal{V}$ then checks the proof with the public statement and accepts or rejects it accordingly.

While early practical deployments of zk-SNARKs were adopted primarily in blockchain systems [11, 63, 115], emerging applications are pushing them into new domains with substantially greater computational demands. Notable examples include zero-knowledge machine learning (ZKML), which addresses trust and privacy concerns in datacenter-outsourced ML by enabling proofs that training or inference was performed correctly without revealing model parameters, training data, or client inputs [18, 32, 80, 117]; and zero-knowledge virtual machines (ZKVMs), which extend verifiable computing to arbitrary general-purpose RAM programs while hiding private inputs and computation details [7, 56, 128].

Modern ZK proving systems differ substantially in their dominant computational kernels. For example, pairing-based zk-SNARKs

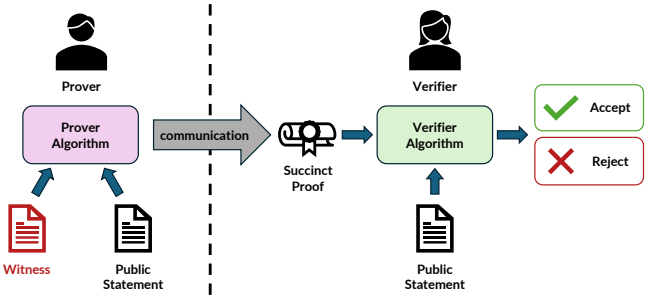


Figure 1: High-level diagram of a non-interactive ZKP

such as Groth16 [37] rely heavily on number-theoretic transforms (NTTs) and multi-scalar multiplication (MSM), and are attractive for their succinct proofs and fast verification. In contrast, a growing class of zk-SNARKs built around the sumcheck protocol [83] offers improved asymptotic prover complexity in many settings. In particular, while NTT-based approaches typically incur quasi-linear prover work, sumcheck-based constructions enable linear prover work, and often with favorable constant factors in practice, yielding a fundamental asymptotic improvement. Accordingly, several modern systems adopt sumcheck as a central component [13, 17, 20, 107, 108, 111, 116, 126]. Across these systems, proof generation is typically the dominant bottleneck, driving significant interest in hardware acceleration.

In particular, in sumcheck-based systems, the sumcheck phase often dominates prover time. Our analysis in Section 4.1 shows that sumcheck accounts for up to 76% of prover runtime and reveals that execution is fundamentally memory-bound, exhibiting high memory bandwidth utilization but with underutilized compute resources. This characteristic is inherent to the memory-intensive nature of the sumcheck protocol. This memory-centric bottleneck stands in contrast to NTT and MSM kernels. Consequently, compute-centric strategies are fundamentally mismatched to sumcheck's computational profile.

Prior work in ZKP hardware acceleration has largely targeted NTT and MSM, employing compute-centric architectures such as GPUs, FPGAs, and custom ASICs to accelerate these compute-bound operations [24, 48, 84, 129, 130]. Recent hardware acceleration works on sumcheck-based protocols achieved notable improvements, but continue to rely on compute-centric designs that do not directly address sumcheck's memory-bound nature [23, 81, 103]. Meanwhile, processing-in-memory (PIM) architectures have demonstrated significant benefits for memory-bound workloads in domains such as machine learning and graph processing [8, 39, 54, 55, 96, 122]. However, their application to ZKP workloads remains unexplored as traditional ZKP protocols based on NTT and MSM are compute-bound. Nevertheless, PIM architectures can be a promising approach for accelerating sumcheck-based ZKP workloads since sumcheck protocols are inherently memory-bound.

In this paper, we propose SumcheckPIM, a PIM architecture optimized for the non-interactive sumcheck protocol. SumcheckPIM enables in-memory processing of sumcheck operations by leveraging near-bank processing engines on DRAM dies and processing units implemented on the logic die of a high-bandwidth memory (HBM) stack. We design a PIM-friendly execution flow of the non-interactive sumcheck protocol that is tailored to the proposed PIM architecture. Furthermore, we present an efficient control scheme that can reduce the DRAM command overhead significantly by exploiting the inherent bank-level data exchange mechanism. We implement the proposed SumcheckPIM architecture on a cycle-accurate DRAM simulator. We estimate the energy and cost overhead by synthesizing the RTL models of processing units. Our evaluation results exhibit that SumcheckPIM is an efficient acceleration architecture for the sumcheck protocol, achieving 5.3×–13.3× and 334.2×–361.1× speedup over GPU and CPU baselines, respectively. To the best of our knowledge, SumcheckPIM is the first study on accelerating the sumcheck protocol using a PIM architecture.

Our key contributions are as follows:

- We propose *SumcheckPIM*, the first HBM-based PIM architecture that accelerates the sumcheck protocol, motivated by profiling results revealing that sumcheck is fundamentally memory-bound with underutilized compute resources.
- We present near-bank processing engines with scalar *256-bit* modular arithmetic (Barrett modular multiplier/adder) to natively support practical field sizes while matching HBM's 32-byte burst data access granularity.
- To handle inter-bank accumulation and Fiat-Shamir challenge generation, we design a *Fiat-Shamir unit* on the logic die, and further add an *inter-bank processing engine* to enable inter-bank folding without host mediation.
- We propose a *DRAM-aware folding scheme* that improves row-buffer locality via row packing and bank ping-ponging, reducing row activations and accelerating end-to-end sumcheck execution.
- We evaluate SumcheckPIM with a cycle-accurate DRAMSim3-based simulator and synthesize SystemVerilog-based RTL, demonstrating substantial speedup and energy savings over CPU/GPU baselines with modest logic-die area overhead.

2 Background

2.1 Zero-knowledge proof

Zero-knowledge proof (ZKP) allows a prover ($\mathcal{P}$) to convince a verifier ($\mathcal{V}$) that a given statement is true without revealing any information beyond the validity of the statement [36]. ZKPs can be expressed as $\mathcal{P}$ proving the computation of $C(x, w) = 0$, where C encodes the computation to be proved (e.g., a circuit), x is the *public* input held by both parties (i.e., *instance*) and w is the *private* input held by $\mathcal{P}$ (i.e., *witness*). A valid proof convinces $\mathcal{V}$ that $\mathcal{P}$ knows such a w while revealing nothing beyond this fact.

ZKP protocols are commonly classified by interactivity. *Interactive* protocols proceed over multiple communication rounds between $\mathcal{P}$ and $\mathcal{V}$, and are therefore round-bound, often with lower per-party computation [118]. In contrast, *non-interactive* ZKPs (NIZKs) eliminate interaction: $\mathcal{P}$ produces a single proof that $\mathcal{V}$ can verify offline [14]. Among NIZKs, zk-SNARKs further offer succinct proofs and low verification overhead, thus zk-SNARKs are the most widely deployed and mature in practice [11, 63, 115].

In zk-SNARK systems, proof generation is a dominant performance hurdle. The classical proof generation involves three computation steps: arithmetization, a polynomial interactive oracle proof

(PIOP), and a polynomial commitment scheme (PCS). Arithmetization maps the target computation into a constraint system over a finite field. PIOP then reduces the satisfiability of these constraints to the verification of a set of polynomial identities driven by $\mathcal{V}$'s uniform challenges, which can be made non-interactive via the Fiat-Shamir transformation [33]. Finally, PCS enables $\mathcal{P}$ to commit to the relevant polynomials and later opens them at specific points to allow $\mathcal{V}$ to check the required relations succinctly.

Arithmetization transforms arbitrary computations into algebraic relations suitable for ZKP, known as a constraint system. The constraint system is a collection of polynomial equations over a finite field that $\mathcal{P}$ aims to satisfy. One of the widely used constraint systems is a rank-1 constraint system (R1CS), where a list of vector triples $\vec{a}, \vec{b}, \vec{c}$ represents the equation system to be satisfied [12]. R1CS satisfiability is given as finding the assignment vector $\vec{z}$ that satisfies $\langle \vec{a}_i, \vec{z} \rangle \cdot \langle \vec{b}_i, \vec{z} \rangle - \langle \vec{c}_i, \vec{z} \rangle = 0$. The constraint size is defined as the total number of constraints in the system. Supporting a larger constraint size enables proving more complex computations.

2.2 Sumcheck protocol

The sumcheck protocol [83] is an interactive proof protocol whose prover runs in $O(n)$ complexity, offering a fundamental asymptotic improvement over the $O(n \log n)$ prover complexity of NTT-based ZKP protocols. Although the original sumcheck is an interactive protocol, it can be represented in a hardware-friendly *non-interactive* manner. The sumcheck protocol is employed in modern ZKP protocols due to its lower complexity and flexibility [13, 17, 20, 107, 108, 111, 116, 126].

In the sumcheck protocol, $\mathcal{P}$ aims to prove to $\mathcal{V}$ that a multilinear polynomial, a multivariate polynomial with degree at most one in each variable, sums to a claimed value K over the Boolean hypercube. More concretely, given a multilinear polynomial $p(x_1, \ldots, x_n)$, $\mathcal{P}$ aims to prove:

$$\sum_{b_1,\ldots,b_n \in \{0,1\}} p(b_1, \ldots, b_n) = K \tag{1}$$

In each round i, $\mathcal{P}$ defines a univariate polynomial $g_i(X)$ representing the partial sum over the remaining variables after fixing the first $i-1$ variables:

$$g_i(X) = \sum_{b_{i+1},\ldots,b_n \in \{0,1\}} p(r_1, \ldots, r_{i-1}, X, b_{i+1}, \ldots, b_n) \tag{2}$$

where $r_1, \ldots, r_{i-1}$ are random challenges sent by $\mathcal{V}$.

Once $\mathcal{V}$ receives univariate polynomial $g_i(X)$, $\mathcal{V}$ checks whether the following conditions hold, for the first round and the i-th round, respectively:

$$g_1(0) + g_1(1) = K \tag{3}$$

$$g_i(0) + g_i(1) = g_{i-1}(r_{i-1}) \tag{4}$$

After n rounds, once all variables are bound, $\mathcal{V}$ checks the final equality:

$$g_n(r_n) = p(r_1, \ldots, r_n) \tag{5}$$

By default, the sumcheck protocol is executed interactively, requiring back-and-forth communication between $\mathcal{P}$ and $\mathcal{V}$ per round. Non-interactive instantiation of the sumcheck protocol can be constructed through the Fiat-Shamir transformation. With the Fiat-Shamir transformation, each challenge r_i is generated locally by $\mathcal{P}$ based on Equation 6, where $H(\cdot)$ is a cryptographic hash function. Since H is deterministic, $\mathcal{V}$ can recompute random challenges and verify consistency, eliminating interactive communication. Non-interactivity has multiple desirable properties, such as allowing multiple parties to verify without having participated in the proof procedure and proof reusability [38, 105]. It is of particular interest from the hardware perspective because eliminating the communication latency bottleneck enables hardware-centric optimization that significantly speeds up end-to-end ZKP generation.

Algorithm 1 Non-interactive sumcheck protocol

Input: Multi-linear polynomial $p(x_1, \ldots, x_n)$ with evaluation table $T[0 : 2^n - 1]$ where $T[\mathbf{b}] = p(b_1, \ldots, b_n)$ for bit-vector $\mathbf{b}$. Cryptographic hash function $H(\cdot)$.

Output: Proof $\pi = \{(g_0^{(1)}, g_1^{(1)}), \ldots, (g_0^{(n)}, g_1^{(n)})\}$ and challenges $r_1, \ldots, r_n$.

$\pi \leftarrow \emptyset$
for round $\leftarrow 1$ to n **do**
  $g_0 \leftarrow 0$
  $g_1 \leftarrow 0$
  halfSize $\leftarrow 2^{n-\text{round}}$
  for idx $\leftarrow 0$ to halfSize $- 1$ **do** ▷ accumulation
    $g_0 \leftarrow g_0 + T[\text{idx}]$
    $g_1 \leftarrow g_1 + T[\text{idx} + \text{halfSize}]$
  end for
  $\pi \leftarrow \pi \cup \{(g_0, g_1)\}$
  $r_{\text{round}} \leftarrow H(\pi)$ ▷ Fiat-Shamir challenge
  for idx $\leftarrow 0$ to halfSize $- 1$ **do** ▷ fold table
    $T[\text{idx}] \leftarrow (1 - r_{\text{round}}) \cdot T[\text{idx}] + r_{\text{round}} \cdot T[\text{idx} + \text{halfSize}]$
  end for
end for
return π, $(r_1, \ldots, r_n)$

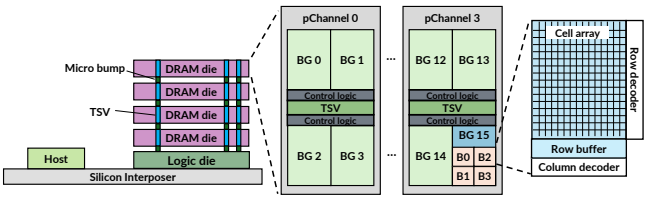


Figure 2: HBM organization

$$r_j = H(\text{proof transcript up to round } j) \tag{6}$$

Algorithm 1 presents the linear-time, linear-space sumcheck algorithm [19, 83], which achieves optimal prover complexity. The algorithm operates on an evaluation table T with size equal to the constraint size 2^n. Each round performs accumulation, random challenge generation, and folding. Folding halves the evaluation table size, where each pair of elements is combined to produce a single element and updates the evaluation table.

2.3 High Bandwidth Memory

High Bandwidth Memory (HBM) is a 3D-stacked DRAM architecture designed to overcome the bandwidth and energy limitations of

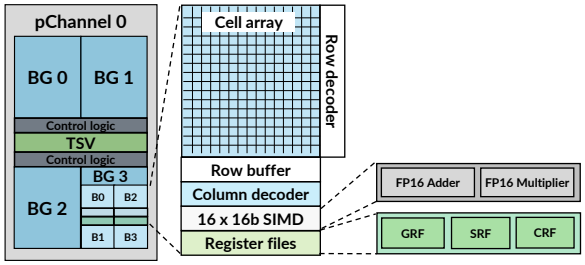


Figure 3: HBM-based PIM architecture

conventional off-chip memory systems. By leveraging a die stacking technology, HBM achieves substantially higher memory bandwidth and improved energy efficiency compared to traditional DRAM interfaces [15, 16, 51, 67–70, 85, 94, 98].

As shown in the leftmost illustration of Figure 2, HBM vertically stacks multiple DRAM dies on top of a logic die by connecting them using through-silicon vias (TSVs) to enable high-speed and wide internal data transfers between memory layers [58, 61]. The stacked DRAM dies are bonded using technologies such as microbumps or hybrid bonding. The logic die, also referred to as the buffer die, integrates I/O drivers, buffers, and physical layer (PHY) circuitry to interface with a host processor. Prior studies have observed that the logic die contains additional area that can accommodate active logic other than conventional circuitry, enabling integration of custom ASIC beyond standard I/O functions [52, 87]. Communication between the HBM stack and the host processor is facilitated through silicon interposer or bridge technologies, such as Intel EMIB [29] or TSMC CoWoS-L [123].

HBM adopts a hierarchical organization to expose high degrees of parallelism. As illustrated in the center of Figure 2, HBM is divided into multiple independent channels, each providing a 128-bit wide data interface. Each channel is further partitioned into two 64-bit pseudo-channels (pCh), enabling finer-grained parallel access. Within each pseudo-channel, memory is organized into bank groups and banks, which operate independently and can process memory commands concurrently. At the bank level, data is stored in rows within DRAM sub-arrays. When a row is accessed, its contents are loaded into the row buffer through a row *activation* operation, incurring a latency of t_{RCD}. Subsequent accesses to the same activated row, known as *row hits*, avoid repeated activations and incur lower latency. Achieving high row buffer locality, maximizing row hits, is a key performance characteristic of HBM and plays an important role in enabling efficient memory access and in-memory computation.

2.4 HBM-based PIM architecture

HBM-PIM architectures enhance HBM by integrating lightweight compute units within the memory system, enabling computation to be performed directly on data stored in DRAM and thereby accelerating memory-bound workloads [60, 66, 72]. Amongst several PIM implementations as discussed in Section 3, our proposed PIM architecture is based on the commodity HBM-based PIM architecture [60]. The commodity HBM-PIM depicted in Figure 3 is based on the HBM2 standard and augments conventional DRAM banks with compute units placed at the bank I/O boundary. To accommodate processing units while maintaining physical compatibility and DRAM timing constraints, half of the DRAM sub-arrays are removed, and each bank is equipped with a 16-lane 16-bit SIMD floating-point unit (FPU) and dedicated register files. By executing arithmetic operations directly near banks, HBM-PIM significantly reduces data movement between the host processor and memory, while exposing the high internal bandwidth of DRAM banks to computation.

HBM-PIM supports three operation modes: single bank (SB) mode, all bank (AB) mode, and all bank PIM (AB-PIM) mode. SB mode corresponds to standard HBM operations, where each memory command targets a single bank. In AB mode, a single column command is broadcast to all banks within a pseudo-channel, enabling lock-step access to the same row and column addresses across banks. AB-PIM mode extends AB mode by allowing column commands to simultaneously trigger PIM instruction executions to enable synchronized in-memory computations across banks and substantially higher effective bandwidth for PIM workloads.

For PIM operations, PIM instructions are stored in a command register file (CRF). Each column command deterministically invokes an instruction from the CRF, tightly coupling computation with DRAM access timing while avoiding unnecessary data transfers through the global I/O interface. Each PIM execution unit includes a 16-wide SIMD datapath supporting FP16 arithmetic, a general register file (GRF) for vector operands and accumulation, and a scalar register file (SRF) for broadcasting scalar values across SIMD lanes. The supported instruction set includes arithmetic operations such as `ADD`, `MUL`, and `MAC`, as well as control and data-movement instructions. Existing industrial HBM-PIM designs primarily target general matrix-vector multiplication (GeMV) and related kernels, which are representative of memory-bandwidth-bound workloads in machine learning and high-performance computing.

To support transitions between SB, AB, and AB-PIM modes without modifying the host processor or accessing privileged DRAM mode registers, HBM-PIM encodes mode configuration using memory-mapped configuration registers. Reserved address regions, referred to as *PIM CONF*, are mapped to internal registers such as the AB mode register (ABMR), SB mode register (SBMR), and a PIM operation mode register. Mode transitions are triggered by issuing ordered sequences of standard DRAM commands to these addresses, avoiding mode register set (MRS) commands and the associated kernel-level overhead.

3 Related work

Processing-in-memory (PIM) and near-data-processing (NDP) architectures have emerged as promising approaches that can mitigate the high cost of data movement in modern systems. Companies such as Samsung [60, 66, 72], SK Hynix [40, 64, 65, 73], UPMEM [26, 95], Facebook [57], and Alibaba [91] have developed industrial PIM/NDP solutions, demonstrating the potential of near or in-memory computing for commercial applications. In the research field, PIM/NDP architectures leveraging 3D-stacked memory technologies such as HMC and HBM have been extensively explored [4–6, 8, 30, 50, 54, 88, 100, 110, 119, 125, 127, 133–135], particularly for accelerating AI workloads.

Previous research has looked at placing custom ASICs on the logic die of HBM and HMC to target a range of memory-intensive

workloads such as graph processing, neural network inference, database operations, and more [4, 5, 30, 34, 50, 88, 99, 100, 102, 110, 119, 125, 127, 133–135]. Numerous studies have built on HBM-PIM for accelerating memory-bound workloads [8, 39, 47, 54, 55, 62, 74, 106, 122, 132]. Additionally, hybrid PIM/NDP architectures that combine in-bank processing units with logic-die accelerators have been proposed to process more complex computational kernels [41, 42, 46, 96, 124]. These designs reduce or eliminate host communication overhead, create a compute hierarchy between bank-level and logic-die processing, and enable placing hardware units with area requirements exceeding constraints of in-bank integration.

Additionally, there have been numerous works that utilized PIM, NDP, and also compute-in-memory (CiM) architectures for targeting cryptography or confidential computing workloads with modular arithmetic as its major primitive, such as fully homomorphic encryption (FHE), multi-party computation (MPC), and post-quantum cryptography (PQC) [27, 28, 35, 53, 59, 75, 78, 89, 90, 97, 120, 131].

Research into hardware acceleration of zero-knowledge proofs has been a growing area. The majority of works on ZKP hardware acceleration target NTT and MSM operations using ASICs [24, 112, 121, 129], GPUs [48, 49, 76, 82, 84], FPGAs [43, 101, 130], and distributed systems [79, 113]. There is a limited number of hardware accelerations of sumcheck-based ZKP protocols [22, 23, 81, 103].

BatchZK [81] demonstrates GPU-based hardware acceleration of the sumcheck-based ZKP protocol. BatchZK implements the core modules of sumcheck-based ZKP protocols, including the sumcheck prover, Spielman encoder, and Merkle tree. BatchZK optimizes for proof throughput using a pipelined GPU execution and dynamic loading method that also achieves high utilization. However, BatchZK trades off latency for throughput, exhibiting a 7.7× increased latency over the ICICLE ZKP GPU [44] baseline on the sumcheck kernel at constraint size 2^{20}.

NoCap [103] is an ASIC accelerator targeting Spartan + Orion. It features vector lanes of processing elements interconnected via a Beneš network, alongside dedicated vector hash and NTT units. NoCap employs a decoupled data-compute architecture with static scheduling and uses an 8MB register file with orchestrated memory accesses to hide memory latency. Sumcheck is mapped to the modular arithmetic vector lanes and shuffle units. NoCap is based on Goldilocks-64, which allows the use of 64-bit modular arithmetic units but requires running sumcheck three times to achieve sufficient security. NoCap achieves 586× and 41× over high-performance CPU and PipeZK [129], a ZKP hardware accelerator, respectively.

4 Motivation

In this section, we analyze the characteristics of the sumcheck protocol using high-performance CPU and GPU systems. Then, we describe the challenges of employing existing PIM architectures to accelerate the sumcheck protocol.

4.1 Characteristics of sumcheck

Long execution time: As described in Section 2.2, the sumcheck-based ZKP prover exhibits the lower complexity of $O(n)$ compared to NTT-based ZKPs. However, the computation of the sumcheck

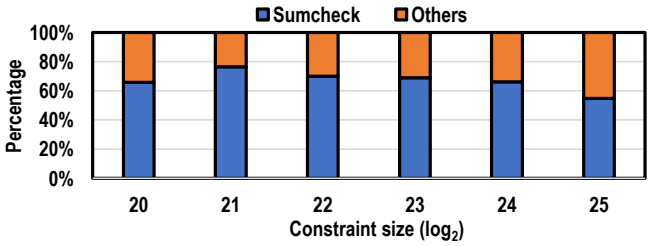


Figure 4: Sumcheck runtime breakdown on CPU

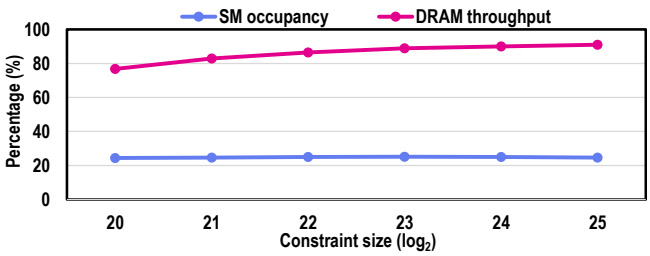


Figure 5: Memory throughput and SM occupancy on GPU

algorithm is still a significant burden in ZKP systems. In order to analyze the performance burden of sumcheck, we profile the execution time of the sumcheck-based Spartan2 library [107] with the Hyrax PCS in various constraint sizes. We use the CPU-based ZKP system configured as listed in Table 2. Figure 4 exhibits the breakdown of the execution time by various constraint sizes. Our analysis reveals that sumcheck occupies 54%–76% of the total execution time of Spartan ZKP. Prior studies also report that sumcheck takes up a large fraction of ZKP execution time. NoCap presents that sumcheck occupies 70% of the total execution time of the Spartan + Orion ZKP protocol [103]. Hyperplonk reports that sumcheck-dominated multi-linear evaluations occupy 61% of total ZKP execution time [17].

Low arithmetic intensity due to weak locality: The sumcheck protocol is inherently memory-bound, and thus it exhibits extremely low arithmetic intensity. Each round performs global accumulation and binary folding that halves the working set data. During the folding operations, the sumcheck kernel reads entire elements in a working set to perform a few field multiplications and additions per folding, then updates the working set accordingly. This process resembles binary tree processing. However, whereas the standard binary tree reductions can exploit spatial locality with depth-first processing, the folding operation in sumcheck exhibits strict inter-round dependencies that enforce breadth-first processing over a large working set, yielding extremely weak spatial locality. Consequently, the sumcheck kernel creates heavy off-chip memory traffic. Prior studies also report the memory-intensive characteristics of the sumcheck protocol [23, 81, 103].

To investigate the low processing efficiency due to heavy demand data traffic, as shown in Figure 5, we profile off-chip memory throughput and streaming multiprocessor (SM) occupancy by running the sumcheck kernel provided by the ICICLE 4.0 CUDA ZKP backend [44] on NVIDIA RTX A6000 GPUs. The analysis results exhibit that the average off-chip memory throughput of the sumcheck kernel is extremely high (77%–95%). In contrast, the processing units

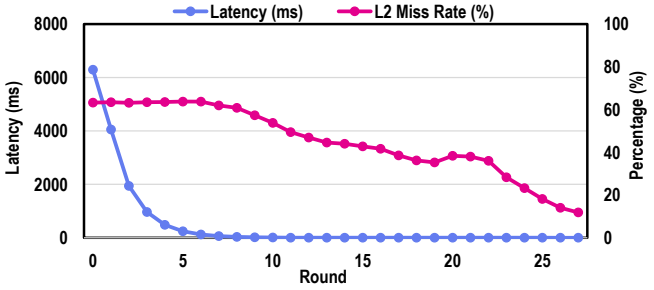


Figure 6: Per-round latency and L2 cache miss rate of sumcheck

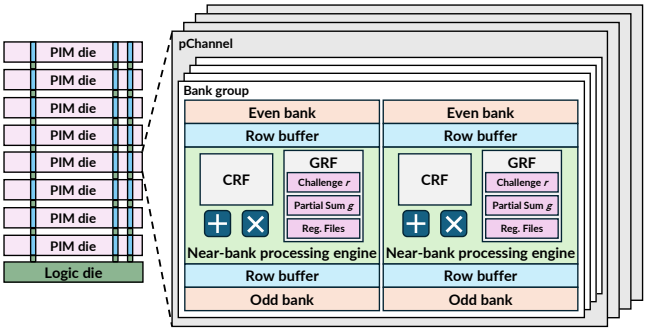


Figure 7: Overview of the SumcheckPIM architecture, focusing on near-bank processing engines

are severely underutilized, with SM occupancy at approximately 25%.

Detailed behavior by sumcheck rounds: To further investigate the behavior of the sumcheck kernel, we profile the execution latency and L2 cache miss rates by sumcheck rounds, as shown in Figure 6. We use the CPU systems listed in Table 2 with the PAPI interface [45] to collect the profiled data. Note that the ICICLE ZKP GPU backend is closed-source, and thus, we utilize the CPU system instead. The sumcheck kernel is bound to a single socket to eliminate non-uniform memory access (NUMA) effects. Our analysis reveals that in early rounds, L2 cache miss rates are high (over 60%) because sumcheck's evaluation table T exceeds the L2 cache capacity. Since sumcheck halves the working set each round, an increasing fraction of the evaluation table fits within the cache hierarchy, improving cache utilization in later rounds. However, our analysis also reveals that the first 5 rounds of sumcheck, which exhibit high cache miss rates, occupy 97% of the total execution time. This indicates that the sumcheck kernel is *significantly memory-bound*, thus we need efficient approaches that can handle such heavy off-chip data traffic by sumcheck.

4.2 Challenges of employing PIM

Since PIM architectures are emerging solutions that can address the high off-chip data movement cost by memory-bound workloads, employing PIM for the sumcheck kernel can be an effective approach that improves the performance of sumcheck-based ZKPs. However, there are several challenges to employing PIM architectures directly for accelerating the sumcheck protocol.

Large-bitwidth modular arithmetic: The sumcheck protocol requires large-bitwidth modular arithmetic to achieve sufficient security levels [10, 37]. In practice, sumcheck-based ZKP systems use large prime fields, with field sizes extending up to 256-bits [17, 107]. However, the processing engines embedded in commodity and academic PIM solutions usually support low-precision data formats such as FP16 and MX8 [62, 96]. Even though optimizations allow to deploy lower-bitwidth operands supported by general-purpose processors [9, 59], these approaches require complex control flows and/or data concatenations, which do not fit well into PIM architectures.

Inter-bank data communications: In order to maximize the processing efficiency on existing memory cell structures, the commodity HBM-based PIM architectures only support data use within a bank or a bank group. Hence, a host processor needs to create abundant off-chip memory accesses for data fetch and write-back to compute data across different banks or bank groups. Such inter-bank computations significantly curtail the performance benefits of PIM architectures. The sumcheck protocol contains several operations that require global data reductions. For instance, the accumulation of all current working set data (Algorithm 1, lines 7 and 8) incurs data reduction across multiple banks. The Fiat-Shamir challenge requires the generated random challenge element to be broadcast to all processing engines (Algorithm 1, line 11). Although the folding scheme (Algorithm 1, line 13) initially may not require any inter-bank data communications in the earlier rounds, the data across multiple banks need to be merged further once the data within a bank are fully reduced.

5 SumcheckPIM

In this work, we propose *SumcheckPIM*, an efficient HBM-based PIM architecture tailored to supporting the non-interactive sumcheck protocol. As shown in Figure 7, SumcheckPIM is implemented based on the HBM2 stack configuration that includes eight DRAM (PIM) dies and one logic die. To support the arithmetic for the sumcheck protocol in PIM, SumcheckPIM includes near-bank processing engines (NBPs) on the PIM dies. Like the commodity HBM-based PIM solution, a single near-bank processing engine is associated with a pair of odd and even banks. In addition, SumcheckPIM implements processing units for the Fiat-Shamir transformation and inter-bank computations on the logic die. Note that the non-interactive sumcheck protocol requires accumulations and next-round challenge generations for the entire working set data, as mentioned in Section 2.2. The processing units on the logic die support computations spanning multiple banks, which cannot be performed by NBPs. SumcheckPIM further optimizes the PIM architecture to support the folding operation efficiently by exploiting DRAM peripheral circuits and the even-odd bank structure of HBM.

To support the large-bitwidth modular arithmetic demanded by ZKP applications, SumcheckPIM exploits the 32-byte (i.e., 256-bit) burst access granularity of HBM. Namely, SumcheckPIM implements 256-bit modular multipliers and adders that operate seamlessly using the basic DRAM commands. By exploiting the wide access granularity of the HBM burst mode and the high-bitwidth arithmetic units implemented in HBM dies, SumcheckPIM can support a wide range of sumcheck-based ZKP applications using the PIM operations solely.

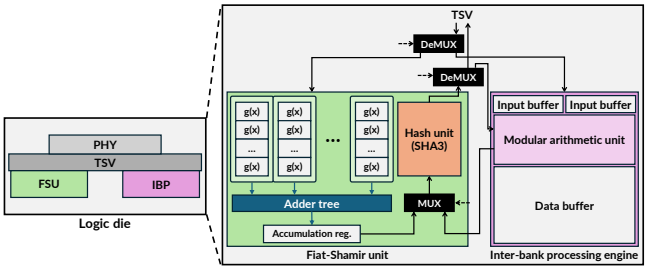


Figure 8: Processing units on the logic die of SumcheckPIM

5.1 Proposed architecture

Near-bank processing engine (NBP): As shown in Figure 7, a single NBP is associated with a pair of odd and even banks to perform field arithmetic operations near the banks. Each NBP includes a 256-bit Barrett modular multiplier, a 256-bit modular adder, and a control unit. The general register file (GRF) within an NBP includes 16 registers with a 256-bit width. GRF is used for holding field elements, random challenge elements, and partial accumulation results. SumcheckPIM supports the data transfers between GRF and the row buffers of odd and even banks using `MOV` and `FILL` instructions. The target bank is selected by the bank select bit allocated in an unused bit field of a DRAM read command during AB-PIM mode.

Fiat-Shamir unit (FSU): An FSU is placed on the logic die to process global accumulations and random challenge generations. As shown in Figure 8, the FSU consists of operand queues, an adder tree, an accumulation register file (ARF) and a SHA3 hash unit. The field arithmetic results stored in banks are delivered to the operand queues through TSVs. The operands transferred from the multiple banks are accumulated by the pipelined adder tree composed of 256-bit modular adders, and then the output of the adder tree is stored in ARF. SumcheckPIM includes a total of 31 modular adders within an adder tree under the configuration of eight PIM dies. The SHA3 hash unit produces random challenge field elements from the accumulated results. The output of the SHA3 hash unit can be routed either to NBPs through TSVs or to the IBP on the logic die.

Inter-bank processing engine (IBP): The IBP on the logic die performs field multiplications and additions further by gathering the results from multiple NBPs. It is required for folding between banks when each of the banks has been fully reduced in the later rounds. The IBP is composed of two input buffers, where each input buffer includes four 256-bit entries, a modular arithmetic unit (a 256-bit Barrett modular multiplier and a 256-bit modular adder), and a 15 KB data buffer for storing the working set data, intermediate values, and proof transcript. As shown in Figure 8, the IBP can also receive the random challenge elements from the FSU and provide the accumulated results back to the SHA3 hash unit.

To support the operations on the logic die (i.e., FSU and IBP), SumcheckPIM includes additional configuration registers, `LOGIC_DIE_MODE` and `LOGIC_DIE_SELECT`, mapped to *PIM CONF* reserved address space. Once `LOGIC_DIE_MODE` is toggled on, DRAM read commands operate similarly to the single-bank (SB) mode, but the data read from a target bank is transferred to the processing units on the logic die. `LOGIC_DIE_SELECT` selects the target processing unit (i.e., IBP or FSU) of the transferred data. Note that `LOGIC_DIE_SELECT` is ignored when `LOGIC_DIE_MODE` is disabled. Our approach bypasses expensive kernel-level MRS accesses [60].

5.2 Instruction set and command interface

SumcheckPIM executes the near-bank PIM instructions in a manner similar to the commodity HBM-based PIM [60]. Namely, SumcheckPIM executes PIM instructions for NBPs by triggering the instructions in the command register file (CRF) with DRAM read commands during AB-PIM mode. All PIM instructions are encoded in a 32-bit command space. SumcheckPIM includes a PIM instruction set for supporting modular arithmetic operations as listed in Table 1a. `MOV` and `FILL` move 32-byte column data between the row buffer and the registers in GRF. `M.ADD`, `M.MULT`, and `M.SUB` are modular instructions of addition, multiplication, and subtraction, respectively. These modular instructions use data in GRF as operands. To execute modular instructions with NBPs, SumcheckPIM loads data from the row buffer into registers using `MOV`, then executes modular arithmetic instructions. Once the requested modular computations are complete, SumcheckPIM stores the results in GRF back to the row buffer using `FILL`.

SumcheckPIM controls the processing units on the logic die by exploiting a DRAM row command interface issued by the host memory controller. Unlike the PIM instructions for NBPs, SumcheckPIM leverages the virtual row address space to encode the commands for IBP and FSU. Note that the number of rows per bank is reduced to half of the original HBM since processing units (i.e., NBPs) occupy a part of the DRAM (PIM) dies. Although the corresponding physical rows do not exist in each bank, their address space remains to support compatibility with the JEDEC standard [46, 66]. Hence, SumcheckPIM can use the most significant bit (MSB) of a row field as a special bit for specifying logic die operations. Specifically, if this bit is set, SumcheckPIM uses the remaining address fields to encode logic die commands. With this approach, SumcheckPIM can control the logic die operations more efficiently without offloading instructions to the logic die.

Table 1: Additional instruction set of SumcheckPIM

(a) PIM instructions for NBPs

Operation	Source A	Source B	Destination
MOV	Row buffer	-	Register
FILL	Register	-	Row buffer
M.ADD	Register	Register	Register
M.SUB	Register	Register	Register
M.MULT	Register	Register	Register

(b) Logic die functions triggered by virtual row commands

Operation	Input	Description
ST	Input buffer index, Data buffer index	Store from input buffer to data buffer
M.ADD	Data buffer index, Data buffer index	Modular addition
M.SUB	Data buffer index, Data buffer index	Modular subtraction
M.MULT	Data buffer index, Data buffer index	Modular multiplication
ACC	ARF Index	Adder tree activation
HASH_WRITE	ARF index or Data buffer index	Push element into hash unit
HASH	-	Process hash
BCAST	-	Broadcast hash result to NBP
IBP.BCAST	Data buffer index	Broadcast hash result to IBP

Table 1b lists the commands for logic die operations. IBP supports the commands for data movement between the data buffer and input buffers as well as modular arithmetic commands (i.e., `M.MULT`, `M.ADD`, and `M.SUB`). FSU is controlled by the commands for accumulation and hash processing. `ACC` activates the adder tree to accumulate values and write into ARF. `HASH_WRITE` forwards

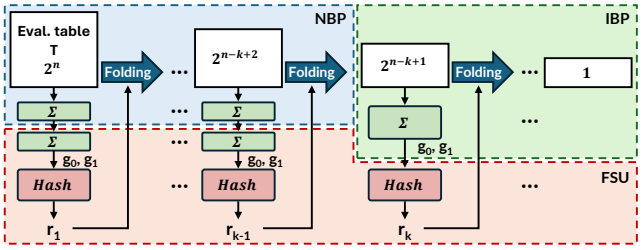


Figure 9: Execution flow of SumcheckPIM

values from IBP's data buffer or ARF to the SHA3 hash unit, and `HASH` triggers hash computation. `BCAST` and `IBP.BCAST` broadcast the hash output to NBPs and IBP, respectively.

5.3 Execution flow

Figure 9 depicts the execution flow of the sumcheck protocol on SumcheckPIM, with shaded regions denoting the processing units involved. The initial evaluation table T is uniformly distributed across banks via round-robin interleaving, first across all even banks, then across all odd banks, to enable folding without inter-bank communication. If the table size exceeds half of the total HBM capacity, it is distributed across both even and odd banks. When it fits within half of the total capacity, SumcheckPIM allocates the evaluation table entirely to either even or odd banks to exploit the DRAM-aware folding scheme described in Section 5.5.

Each round of the sumcheck protocol consists of three phases: *accumulation*, *challenge generation*, and *folding*.

Accumulation: The sumcheck protocol performs the accumulation of the entire working set. SumcheckPIM performs the accumulation phase hierarchically: intra-bank accumulation is handled by each NBP, and inter-bank accumulation is performed by FSU. Namely, each NBP accumulates the working set data within the corresponding bank concurrently, then SumcheckPIM forwards the partial sums generated by NBPs to FSU sequentially via TSV channels. This operation is triggered by DRAM commands issued by the host memory controller. FSU computes the global sum using the adder tree as described in the previous section.

Challenge generation: SumcheckPIM delivers the accumulation results to the SHA3 hash unit, which computes the Fiat-Shamir challenge $r_{round} = H(\pi)$. SumcheckPIM then broadcasts the generated random challenge element (r_{round}) to all NBPs using the `BCAST` logic die command. The random challenge is subsequently written to the GRF of each NBP.

Folding: Each NBP updates the corresponding local portion of the evaluation table using the computed random challenge. Since folding combines pairs of elements within the same bank, all NBPs can perform the folding operation independently in parallel. When each bank includes only a single element in later rounds, SumcheckPIM forwards the element from each bank to IBP to perform the folding operation further by aggregating elements across banks.

5.4 Inter-bank folding

As the sumcheck protocol progresses, the size of the evaluation table is halved in each round. Consequently, the working set within each bank also halves every round until each bank contains only a single element. In order to further perform the folding operation

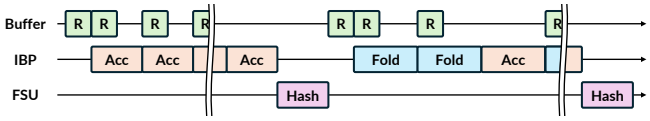


Figure 10: Timing of the inter-bank folding

entirely in memory without host intervention, SumcheckPIM transfers elements from multiple banks to the IBP via TSVs, where they are aggregated. Note that the baseline HBM-based PIM architecture does not support inter-bank communication, and thus the host processor needs to fetch the remaining elements from multiple banks to complete the folding operation. Such host–PIM data transfers incur significant data-movement overheads (i.e., high latency and energy consumption). SumcheckPIM addresses the overheads of the folding operations across banks using simple processing engines on the logic die and the pipelining scheme.

During the inter-bank folding operation (round k and subsequent rounds in Figure 9), the host memory controller issues DRAM commands to transfer the element from each bank to IBP, then IBP performs accumulation across elements from multiple banks. Like the normal challenge-generation phase, FSU computes the random challenge element from the accumulated working set data. Using the `IBP.BCAST` command, the generated random challenge element is stored in the IBP data buffer for the next round of operation. Once the IBP processes up to the final round, the host reads the proof transcript from the IBP data buffer.

Figure 10 exhibits the pipelining of the first round of inter-bank folding. As mentioned in Section 5.3, SumcheckPIM initiates the inter-bank folding when each bank contains only a single element. SumcheckPIM transfers the single element in each bank twice: first to calculate the challenge element (i.e., the first hash) and second to perform the folding operation with the first hash. During the challenge generation, element fetches from banks (denoted as *R* in Figure 10) are overlapped with IBP accumulation (denoted as *Acc*). IBP stores the accumulated value in the data buffer, and FSU hashes this value to generate the challenge element for the next round. Note that the fetched elements are not stored in IBP during this phase. Next, for the folding operation, SumcheckPIM again transfers the single element in each bank to IBP. These element fetches are overlapped with the folding operation. The result of the folding operation, representing the working set, is then saved to the data buffer in IBP. In subsequent rounds, as the working set data now exists in the IBP data buffer, the three steps of the sumcheck round are processed by IBP and FSU without fetching elements from banks. Note that by performing the element fetch twice in the first round of inter-bank folding, SumcheckPIM can employ a smaller data buffer for storing the working set data (i.e., half of the single element fetch case).

5.5 DRAM-aware folding scheme

SumcheckPIM employs a DRAM-aware folding scheme that maximizes row-buffer locality through a row-packing strategy and a ping-pong mechanism between odd and even banks. As described in Section 5.1, each NBP is associated with a pair of adjacent banks (i.e., one odd and one even bank). Accordingly, a single NBP can access the row buffers of both banks by adjusting the source and destination pointers of the `MOV` and `FILL` instructions. By exploiting

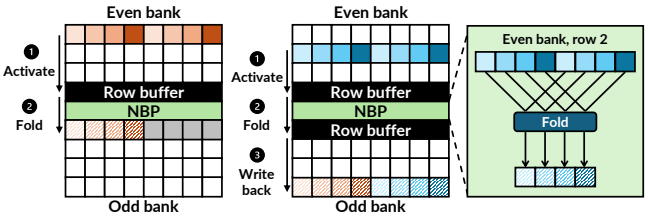


Figure 11: DRAM-aware folding scheme

this unique HBM-based PIM architecture, SumcheckPIM further optimizes the performance of folding operations.

The row-packing scheme prevents row fragmentation by combining folded outputs from multiple rows into a single row, maximizing operations per row activation. If an NBP accesses only a single bank, this row-packing scheme cannot be applied since the row buffer updated with the folded output needs to be written back to the bank before activating (i.e., loading) a new source row for the folding operation. This process is similar to a *row conflict* in conventional DRAM operation, which requires additional commands such as *precharge* and *activation* to replace the row buffer. SumcheckPIM enables row-packing through bank ping-ponging. Namely, SumcheckPIM writes the folded outputs to the opposite bank's row buffer to avoid row conflicts.

Figure 11 depicts the proposed DRAM-aware folding scheme. The leftmost figure shows processing of the first row: the row is activated (loaded) into the even bank's row buffer, and folded elements are written to half of the odd bank's row buffer. In the center figure, the second row of the even bank is activated, and folded elements are written to the remaining half of the odd bank's row buffer. Once two rows of the even bank have been folded and packed into a single row of the odd bank, the row buffer is written back to the odd bank. This process repeats for all remaining rows. In subsequent rounds, the scheme changes the source and destination, folding from the odd bank back into the even bank. By separating the source and destination banks, SumcheckPIM enables row packing without row conflicts. Compared to a naïve single-bank folding approach, our proposed DRAM-aware folding scheme reduces the number of activations by more than half.

For large-scale ZKP workloads, the evaluation table may initially occupy all rows within a bank. If both odd and even banks are fully occupied, the proposed DRAM-aware folding scheme cannot be applied since the destination bank lacks *empty* rows to store folded outputs. In this case, SumcheckPIM performs in-place updates for the first folding round. Namely, the folding operation is performed using one row from the odd bank and one row from the even bank, then the folded outputs are packed into a single row in one of the banks. Note that, after this round, one bank holds the *updated* folded outputs while the other bank contains the *outdated* data. This frees the latter bank to be used as the ping-pong destination, enabling the DRAM-aware folding scheme for the subsequent rounds.

6 Evaluation

6.1 Methodology

We implement the proposed SumcheckPIM architecture using a cycle-accurate PIM simulator modified from DRAMSim3 [77] configured with an HBM2 model as listed in Table 2. We evaluate the end-to-end performance of the sumcheck protocol kernels across a wide range of ZKP workloads that use various constraint sizes. For the baseline ZKP systems, we use GPU and CPU systems as shown in Table 2. We use the ICICLE 4.0 library [44] to run the ZKP workloads on the baseline systems. Note that ICICLE supports optimized multi-threaded sumcheck protocol kernels for both CPU and GPU platforms. We measure the latency of the sumcheck protocol ten times on the real hardware systems to take the average latency of the sumcheck protocol execution.

Table 2: System configurations used in evaluation.

Component	Configuration
SumcheckPIM	
Memory type	HBM2
No. of pChannels	32
No. of BG / pChannel	4
No. of banks / BG	4
No. of DRAM rows	16,384
No. of DRAM columns	32
Clock frequency	1,000 MHz
Timing parameters	$t_{CCDS} = 1$, $t_{CCDL} = 2$, $t_{RAS} = 34$, $t_{RP} = 14$, $t_{RCDRD/WR} = 14$, $t_{RRDS} = 4$, $t_{RRDL} = 6$
GPU (NVIDIA RTX A6000)	
No. of CUDA cores	10752
Clock frequency	1,410 MHz / 1,860 MHz
Memory bandwidth	768 GB/s
Device memory	48 GB GDDR6X
CPU (Intel Xeon Gold 6330, dual socket)	
No. of cores and threads	28×2 / 56×2
Clock frequency	2,000 MHz / 3,100 MHz
Device memory	512 GB DDR4

To compare the energy consumption of SumcheckPIM, we measure the energy consumption of CPU and GPU baseline systems using on-chip power sensor interfaces. We obtain CPU energy consumption through the Intel RAPL interface, which captures energy usage of the cores and the attached DRAM [25]. We collect GPU power traces via the NVML API (`nvmlDeviceGetPowerUsage`) [93] with a sampling interval of 10 ms. For SumcheckPIM, we estimate the energy consumption using the built-in power model of DRAMSim3, which accounts for memory state transitions and their corresponding current draws in accordance with the JEDEC standard. We estimate the energy consumption of compute units by combining cycle counts of each active logic component, including SRAM, with power estimates obtained from Synopsys Design Compiler using the 14 nm SAED FinFET process library. We also estimate the area overhead by synthesizing the RTL models of processing units under the same synthesis setup.

6.2 Workload

We use the workload listed in Table 3 that scales from 2^{20} to 2^{25} to reflect real-world ZKP applications. Batched AES, ECDSA, and Merkle Tree are widely used benchmarks for ZKPs [1–3]. We also consider emerging ZKP applications. ZEN [31] is an optimizing compiler for verifiable zero-knowledge neural network inference. Under ZEN's inference verification scheme, ShallowNet on the

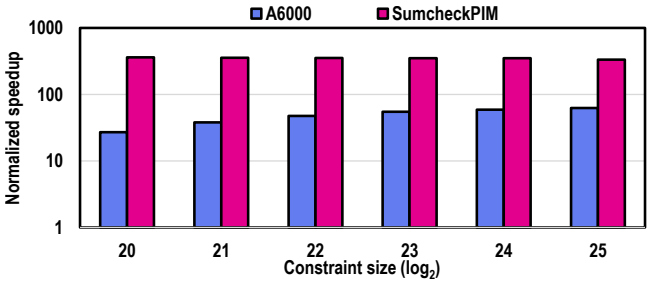


Figure 12: Latency comparison with baselines

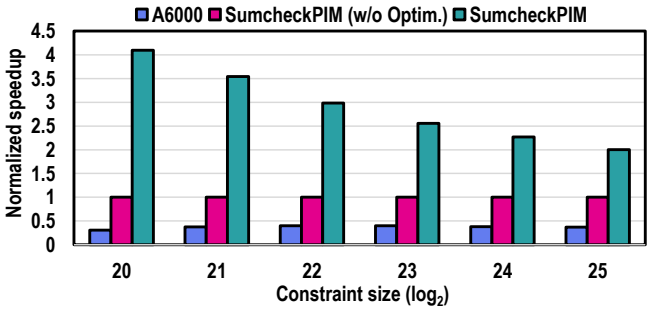


Figure 13: Performance of DRAM-aware folding scheme

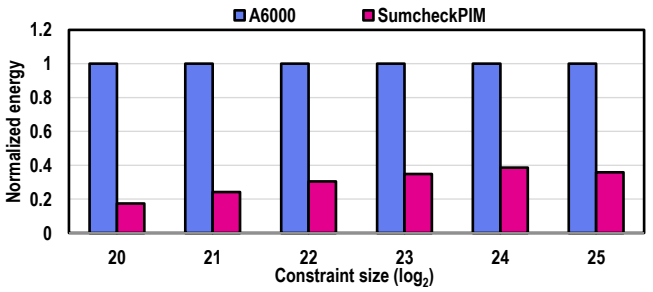


Figure 14: Energy reduction compared to baselines

MNIST dataset and LeNet on the ORL dataset yield 2^{22} and 2^{24} constraints, respectively. LitmusDB [114] is a verifiable database management system that uses ZKP to prove transaction correctness, from which we derive a representative workload yielding 2^{25} constraints. At this largest constraint size, the evaluation table fully utilizes the 4 GB capacity of the SumcheckPIM stack. Previous works have used these selected workloads, and scales are representative of the current state of zero-knowledge proof hardware acceleration [23, 81, 103, 112]. We instantiate the architecture with a BN254 field modulus and R1CS constraint system that will enable comparison with the ICICLE 4.0 library and with prior work.

Table 3: ZKP workloads and associated constraint sizes

Benchmark	Size	Benchmark	Size
Batched AES [3]	2^{20}	Merkle Tree [2]	2^{23}
Verify ECDSA [1]	2^{21}	LeNet [31]	2^{24}
ShallowNet [31]	2^{22}	LitmusDB [114]	2^{25}

6.3 Performance

SumcheckPIM outperforms the baseline systems for all evaluated constraint sizes from 2^{20} to 2^{25} through effective in-memory processing. Figure 12 exhibits the speedup of the GPU and SumcheckPIM normalized to CPU latency. SumcheckPIM achieves 5.3×–13.3× and 334.2×–361.1× over the GPU and CPU baselines, respectively. SumcheckPIM achieves its highest speedup over the GPU baseline at constraint size 2^{20}, and speedup reduces for higher constraint sizes evaluated. We mainly attribute this decrease in speedup for increasing constraint sizes to GPU baselines not fully saturating the DRAM bandwidth for smaller constraint sizes. As depicted in Figure 5, aggregate DRAM throughput profiled with NVIDIA Nsight Compute 2025.1 [92] at 2^{20} shows 76.7% and saturates to 91.0% at 2^{25}.

We also evaluate the performance impact of the proposed DRAM-aware folding scheme. Figure 13 shows the latency of the RTX A6000 GPU and SumcheckPIM with optimization (i.e., DRAM-aware folding), normalized by SumcheckPIM without optimization. Evaluation shows SumcheckPIM with the proposed DRAM-aware folding scheme provides 2.0×–4.1× speedup over SumcheckPIM without optimization. Note that SumcheckPIM without optimization still outperforms the GPU baseline, where the RTX A6000 GPU achieves 0.31×–0.40× speedup with respect to SumcheckPIM without optimization. The DRAM-aware folding scheme steers memory accesses toward fewer rows with data, which improves row buffer locality. Averaged across all constraint sizes, the proposed scheme achieves a row buffer hit rate of 98.9%. The proposed scheme also achieves a significant 72.9% reduction in row activation commands over the SumcheckPIM without optimization on average. A notable trend of Figure 13 is that the speedup of SumcheckPIM with optimization reduces from 4.1× at 2^{20} to 2.0× at 2^{25} constraint sizes compared to SumcheckPIM without optimization. This trend reflects that the naïve scheme is relatively more inefficient at 2^{20} than at 2^{25}. At 2^{20} constraint sizes, the naïve scheme processes early rounds with inter-row folding, which incurs two row activations per folding operation. In contrast, 2^{25} processes the runtime-dominant early rounds with intra-row folding, which has a relatively higher ratio of operations per activation. Consequently, at constraint size 2^{20}, DRAM-aware folding optimization demonstrates a greater speedup.

6.4 Energy efficiency

Figure 14 shows energy consumption normalized to the GPU baseline. The evaluation results show a considerable reduction in energy from 82.6% at 2^{20} to 64.2% at 2^{25} constraint size, demonstrating that SumcheckPIM is an energy-efficient solution for the sumcheck protocol. Energy reduction decreases for higher constraint size as the GPU baseline reduces its relative latency for higher constraint sizes by better saturating the DRAM throughput.

To further demonstrate the efficiency of SumcheckPIM, energy-delay-product (EDP) normalized by the CPU baseline is shown in Figure 15. SumcheckPIM shows 68,477×–80,452× and 14.97×–76.23× EDP improvements over CPU and GPU baselines, respectively. EDP improvements being greater than observed speedup are expected from our design being superior both in terms of latency and energy consumption.

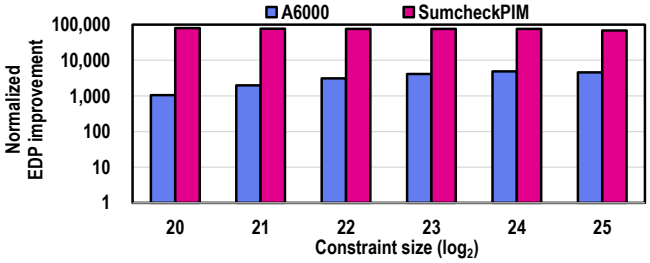


Figure 15: Energy delay product improvement compared to baselines

Table 4: Energy breakdown at 2^{25} constraint size

	DRAM RD/WR	Compute	Other
Energy (%)	97.7	2.12	0.18

We present the energy breakdown of SumcheckPIM at 2^{25} constraint size in Table 4. Note that most energy consumption is from the DRAM read and write, which includes DRAM operations such as activation and precharge, making up 97.7% of total energy consumption. Compute energy consumption makes up second highest portion of 2.12%, which is significantly less than DRAM read and write energy. Other energy consumption reported by DRAMSim3, such as standby energy, makes up the remaining 0.18% of total energy consumption. We note that SumcheckPIM satisfies the power constraint of HBM2.

6.5 Area

We evaluate the hardware area cost of SumcheckPIM units. We synthesize the RTL models of NBP and logic die units using a SAED 14 nm standard-cell library and then scale the results to 20 nm process node using DeepScaleTool [104]. Although DeepScaleTool does not natively support the 20 nm node, we manually compute and apply the corresponding scaling factor to ensure accurate normalization across technologies.

Based on the data in published documents [72, 109], the PIM unit size of the commodity HBM-based PIM system is approximately 0.94 mm^2. The measured area of the NBP, including the modular multiplier, modular adder, and register files introduced by SumcheckPIM, is 0.59 mm^2, which is smaller than the area of a single in-memory processing unit in existing commodity HBM-PIM architectures.

To accurately evaluate the area overhead of SumcheckPIM processing units integrated into the logic die, we quantify their area relative to the logic die area of a conventional HBM2 device. Based on reported measurements, the SumcheckPIM logic die units, comprising the Fiat-Shamir unit, inter-bank processing engine, and control unit, occupy a total area of 9.16 mm^2. Given the reported logic die area of 96 mm^2 for HBM2 devices [21, 71], this corresponds to 9.54% of the total logic die area.

The inter-bank processing engine pipelining optimization described in Section 5.4 also yields significant logic die area savings. An implementation without the proposed optimization would require 28 KB of data buffer on the logic die instead of 15 KB with optimization. This optimization reduces the logic die area overhead by 43% compared to the non-pipelined configuration. Of the total buffer on the logic die, 2.4 KB is used for storing the proof transcript, and the remaining 12.6 KB is allocated for the working set data.

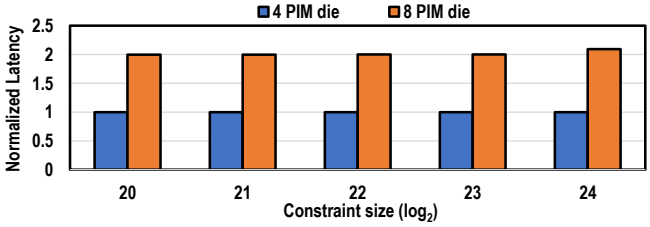


Figure 16: Performance by the number of PIM dies

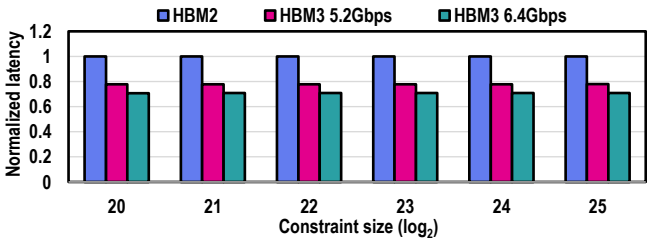


Figure 17: Performance by HBM2 and HBM3 timing parameters

6.6 Sensitivity study

SumcheckPIM is implemented using 32 pseudo-channels of a single HBM stack, which implies that SumcheckPIM utilizes all eight dies of an HBM2 stack as PIM dies. Considering that the commodity HBM-based PIM product uses four PIM dies out of a total of eight DRAM dies [60], we perform a sensitivity study between four and eight PIM die configurations. As shown in Figure 16, we observe that the eight PIM die configuration consistently provides approximately 2× speedup over the four PIM die configuration. This result demonstrates that SumcheckPIM scales with the number of channels and also that global logic die operation has minimal overhead on bank parallel execution. Accordingly, we expect SumcheckPIM to scale further with the larger die stack counts supported by HBM3 and HBM4 standards [86, 87].

Although no commercial products exist to date, several academic studies have proposed HBM-based PIM architecture based on HBM3 [46, 62, 96]. We evaluate the impact of HBM technology on the performance of SumcheckPIM with the results presented in Figure 17. To model the HBM3 configurations, we reference the timing parameters from AttAcc [96] and the JEDEC standard [86]. We select two specific configurations: the 5.2 Gbps configuration used in AttAcc and the 6.4 Gbps variant which represents the peak bandwidth per pin of commercial HBM3. We separately synthesize the logic units using the same procedure described in Section 6.1. We verify them to be functional at the target simulated clock frequency. We observe consistent average latency reductions of 22.3% and 29.1% for the HBM3 5.2 Gbps and 6.4 Gbps configurations, respectively. Note that the measured latency reduction is smaller than the relative increase in clock frequency as timing parameter clock cycles are increased to satisfy internal DRAM timing constraints.

7 Discussion

Comparison with prior work: We compare SumcheckPIM with prior sumcheck accelerators. In the case of BatchZK [81], we do not include a comparison, as it is designed to optimize throughput rather than latency. BatchZK reports a 0.130× speedup over the ICICLE baseline, which SumcheckPIM already surpasses, as demonstrated in Section 6.3. We compare against NoCap [103], a custom ASIC accelerator for Spartan + Orion that assumes 1 TB/s HBM bandwidth. While NoCap does not report sumcheck latency directly, approximately 70% of its runtime is attributed to sumcheck as reported by the authors. Compared to NoCap's reported end-to-end latency for scaled AES and SHA benchmarks (approximately 2^{24} and 2^{25} constraints, respectively), SumcheckPIM achieves 2.13× and 2.09× speedup over NoCap. SumcheckPIM with a four PIM die configuration also surpasses NoCap, delivering a 1.02× speedup at constraint size of 2^{24}.

8 Conclusion

In this paper, we present SumcheckPIM, an HBM-based PIM architecture designed to accelerate the non-interactive sumcheck protocol. Our analysis reveals that the sumcheck protocol is inherently memory-bound, and thus processing units are frequently underutilized despite its simpler computational structure compared to conventional NTT and MSM kernels. In order to address the performance hurdles caused by the high data movement cost of the existing sumcheck protocol, we design an HBM-based PIM architecture that can support the computations of the sumcheck protocol on the base architecture of an HBM stack. SumcheckPIM supports in-memory processing of the sumcheck protocol by combining processing units (PUs) near banks on DRAM dies and dedicated processing engines deployed on the logic die. SumcheckPIM also exploits the data flows via TSVs on an HBM stack to support inter-bank computations. We further develop a PIM-oriented execution flow tightly coupled with the proposed architecture. We also propose a control scheme optimized for the PU architecture associated with odd and even banks on the HBM-based PIM. We implement SumcheckPIM using a cycle-accurate DRAM simulator. Our evaluation results reveal that SumcheckPIM achieves 5.3×–13.3× and 334.2×–361.1× speedup over GPU and CPU baselines, respectively. SumcheckPIM is also an energy-efficient solution, achieving 64.2%–82.6% energy reduction over the GPU baseline.

Acknowledgments

The authors thank Jiaxiang Li and Jingyang Liu for their helpful feedback. The authors also appreciate the anonymous reviewers for their valuable comments. This work was supported in part by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2025-02304483 Chiplet-Based Hub SoC Development Optimized for On-Device Al, RS-2024-00459774 RISC-V Based System Software Development for Open Ecosystem of SDR, IITP-2026-RS-2020-II201819 ICT Creative Consilience Program), and in part by the NATO Science for Peace and Security Programme under Grant G7200. The EDA tool was supported by the IC Design Education Center (IDEC), Korea. Gunjae Koo is the corresponding author.

References

[1] 2022. circom-ecdsa. https://github.com/0xPARC/circom-ecdsa.
[2] 2022. r1cs-tutorial. https://github.com/arkworks-rs/r1cs-tutorial.
[3] 2024. aes-circom. https://github.com/crema-labs/aes-circom.
[4] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. A scalable processing-in-memory accelerator for parallel graph processing. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. 105–117.
[5] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture. *Proceedings of the 42nd Annual International Symposium on Computer Architecture* 43, 3S (2015), 336–348.
[6] Berkin Akin, Franz Franchetti, and James C. Hoe. 2015. Data reorganization in memory using 3D-stacked DRAM. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture* (Portland, Oregon) *(ISCA '15)*. Association for Computing Machinery, New York, NY, USA, 131–143. https://doi.org/10.1145/2749469.2750397
[7] Arasu Arun, Srinath Setty, and Justin Thaler. 2024. Jolt: SNARKs for Virtual Machines via Lookups. In *Advances in Cryptology – EUROCRYPT 2024*, Marc Joye and Gregor Leander (Eds.). Springer Nature Switzerland, Cham, 3–33.
[8] Daehyeon Baek, Soojin Hwang, and Jaehyuk Huh. 2024. pSyncPIM: Partially Synchronous Execution of Sparse Matrix Operations for All-Bank PIM Architectures. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. 354–367. https://doi.org/10.1109/ISCA59077.2024.00034
[9] Suyash Bagad, Yuval Domb, and Justin Thaler. 2024. The Sum-Check Protocol over Fields of Small Characteristic. Cryptology ePrint Archive, Paper 2024/1046. https://eprint.iacr.org/2024/1046
[10] Paulo S. L. M. Barreto and Michael Naehrig. 2006. Pairing-Friendly Elliptic Curves of Prime Order. In *Selected Areas in Cryptography*, Bart Preneel and Stafford Tavares (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 319–331.
[11] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy*. 459–474. https://doi.org/10.1109/SP.2014.36
[12] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. 2013. SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge. In *Advances in Cryptology – CRYPTO 2013*, Ran Canetti and Juan A. Garay (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 90–108.
[13] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. 2018. Aurora: Transparent Succinct Arguments for R1CS. Cryptology ePrint Archive, Paper 2018/828. https://eprint.iacr.org/2018/828
[14] Manuel Blum, Paul Feldman, and Silvio Micali. 1988. Non-interactive zero-knowledge and its applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing* (Chicago, Illinois, USA) *(STOC '88)*. Association for Computing Machinery, New York, NY, USA, 103–112. https://doi.org/10.1145/62212.62222
[15] Kwanyeob Chae, Jiyeon Park, Jaegeun Song, Billy Koo, Jihun Oh, Shinyoung Yi, Won Lee, Dongha Kim, Taekyung Yeo, Kyeongkeun Kang, Sangsoo Park, Eunsu Kim, Sukhyun Jung, Sanghune Park, Sungcheol Park, Mijung Noh, Hyogyuem Rhew, and Jongshin Shin. 2023. A 4nm 1.15TB/s HBM3 Interface with Resistor-Tuned Offset-Calibration and In-Situ Margin-Detection. In *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. 1–3. https://doi.org/10.1109/ISSCC42615.2023.10067736
[16] Kwanyeob Chae, Jaegeun Song, Yoonjae Choi, Jiyeon Park, Billy Koo, Jihun Oh, Shinyoung Yi, Won Lee, Dongha Kim, Kyeongkeun Kang, Eunsu Kim, Juyoung Kim, Sanghune Park, Sungcheol Park, Mijung Noh, Hyo Gyuem Rhew, and Jongshin Shin. 2024. A 4-nm 1.15 TB/s HBM3 Interface With Resistor-Tuned Offset Calibration and In Situ Margin Detection. *IEEE Journal of Solid-State Circuits* 59, 1 (2024), 231–242. https://doi.org/10.1109/JSSC.2023.3330485
[17] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. 2022. HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates. Cryptology ePrint Archive, Paper 2022/1355. https://eprint.iacr.org/2022/1355
[18] Bing-Jyue Chen, Suppakit Waiwitlikhit, Ion Stoica, and Daniel Kang. 2024. ZKML: An Optimizing System for ML Inference in Zero-Knowledge Proofs. In *Proceedings of the Nineteenth European Conference on Computer Systems* (Athens, Greece) *(EuroSys '24)*. Association for Computing Machinery, New York, NY, USA, 560–574. https://doi.org/10.1145/3627703.3650088
[19] Alessandro Chiesa, Elisabetta Fedele, Giacomo Fenzi, and Andrew Zitek-Estrada. 2024. A Time-Space Tradeoff for the Sumcheck Prover. Cryptology ePrint Archive, Paper 2024/524. https://eprint.iacr.org/2024/524
[20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. 2020. Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS. In *Advances in Cryptology – EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I* (Zagreb, Croatia).

Springer-Verlag, Berlin, Heidelberg, 738–768. https://doi.org/10.1007/978-3-030-45721-1_26

[21] Ki Chul Chun, Yong Ki Kim, Yesin Ryu, Jaewon Park, Chi Sung Oh, Young Yong Byun, So Young Kim, Dong Hak Shin, Jun Gyu Lee, Byung-Kyu Ho, Min-Sang Park, Seong-Jin Cho, Seunghan Woo, Byoung Mo Moon, Beomyong Kil, Sungoh Ahn, Jae Hoon Lee, Soo Young Kim, Seouk-Kyu Choi, Jae-Seung Jeong, Sung-Gi Ahn, Jihye Kim, Jun Jin Kong, Kyomin Sohn, Nam Sung Kim, and Jung-Bae Lee. 2021. A 16-GB 640-GB/s HBM2E DRAM With a Data-Bus Window Extension Technique and a Synergetic On-Die ECC Scheme. *IEEE Journal of Solid-State Circuits* 56, 1 (2021), 199–211. https://doi.org/10.1109/JSSC.2020.3027360

[22] Alhad Daftardar, Jianqiao Mo, Joey Ah-kiow, Benedikt Bunz, Siddharth Garg, and Brandon Reagen. 2026. zkPHIRE: A Programmable Accelerator for ZKPs over HIgh-degRee, Expressive Gates . In *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE Computer Society, Los Alamitos, CA, USA, 1–15. https://doi.org/10.1109/HPCA68181.2026.11408480

[23] Alhad Daftardar, Jianqiao Mo, Joey Ah-kiow, Benedikt Bünz, Ramesh Karri, Siddharth Garg, and Brandon Reagen. 2025. Need for zkSpeed: Accelerating HyperPlonk for Zero-Knowledge Proofs. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25)*. Association for Computing Machinery, New York, NY, USA, 1986–2001. https://doi.org/10.1145/3695053.3731021

[24] Alhad Daftardar, Brandon Reagen, and Siddharth Garg. 2024. SZKP: A Scalable Accelerator Architecture for Zero-Knowledge Proofs. In *2024 33rd International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 271–283.

[25] Howard David, Eugene Gorbatov, Ulf R. Hanebutte, Rahul Khanna, and Christian Le. 2010. RAPL: memory power estimation and capping *(ISLPED '10)*. Association for Computing Machinery, New York, NY, USA, 189–194. https://doi.org/10.1145/1840845.1840883

[26] Fabrice Devaux. 2019. The true processing in memory accelerator. In *2019 IEEE Hot Chips 31 Symposium (HCS)*. IEEE Computer Society, 1–24.

[27] Lin Ding, Song Bian, Penggao He, Yan Xu, Gang Qu, and Jiliang Zhang. 2024. APACHE: A Processing-Near-Memory Architecture for Multi-Scheme Fully Homomorphic Encryption. arXiv:2404.15819 [cs.AR] https://arxiv.org/abs/2404.15819

[28] Lin Ding, Song Bian, and Jiliang Zhang. 2025. *PIMA-LPN: Processing-in-Memory Acceleration for Efficient LPN-Based Post-Quantum Cryptography*. IEEE Press, 1–6. https://doi.org/10.1109/DAC56929.2023.10247891

[29] Gang Duan, Yingying Zhang, Andrey Gunawan, Yuxin Fang, Johan Mousavi, Amey Anant Apte, Numair Ahmed, Shruti Sharma, Sid Alur, Anil Chandolu, Xinyu Li, Manni Mo, Jesse Jones, Robin McRee, Rungmai Limvorapitux, Chandra Subramani, Mohammad Rahman, Tarek Ibrahim, Ravi Eluri, Sid Kumar, Sai Agraharam, and Rahul Manepalli. 2025. EMIB-T (TSV) Advanced Packaging Technology EMIB's Next Evolution. In *2025 IEEE 75th Electronic Components and Technology Conference (ECTC)*. 254–258. https://doi.org/10.1109/ECTC51687.2025.00051

[30] Amin Farmahini-Farahani, Jung Ho Ahn, Katherine Morrow, and Nam Sung Kim. 2015. NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 283–295.

[31] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. 2021. ZEN: An Optimizing Compiler for Verifiable, Zero-Knowledge Neural Network Inferences. Cryptology ePrint Archive, Paper 2021/087. https://eprint.iacr.org/2021/087

[32] Boyuan Feng, Zheng Wang, Yuke Wang, Shu Yang, and Yufei Ding. 2024. ZENO: A Type-based Optimization Framework for Zero Knowledge Neural Network Inference. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 450–464. https://doi.org/10.1145/3617232.3624852

[33] Amos Fiat and Adi Shamir. 1987. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology — CRYPTO' 86*, Andrew M. Odlyzko (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 186–194.

[34] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China) *(ASPLOS '17)*. Association for Computing Machinery, New York, NY, USA, 751–764. https://doi.org/10.1145/3037697.3037702

[35] Sahar Ghoflsaz Ghinani, Jingyao Zhang, and Elaheh Sadredini. 2025. Enabling low-cost secure computing on untrusted in-memory architectures. In *Proceedings of the 34th USENIX Conference on Security Symposium* (Seattle, WA, USA) *(SEC '25)*. USENIX Association, USA, Article 91, 19 pages.

[36] S Goldwasser, S Micali, and C Rackoff. 1985. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing* (Providence, Rhode Island, USA) *(STOC '85)*. Association for Computing Machinery, New York, NY, USA, 291–304. https://doi.org/10.1145/22145.22178

[37] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. In *Proceedings, Part II, of the 35th Annual International Conference on Advances in Cryptology — EUROCRYPT 2016 - Volume 9666*. Springer-Verlag, Berlin, Heidelberg, 305–326.

[38] Jens Groth, Rafail Ostrovsky, and Amit Sahai. 2006. Perfect non-interactive zero knowledge for NP. In *Proceedings of the 24th Annual International Conference on The Theory and Applications of Cryptographic Techniques* (St. Petersburg, Russia) *(EUROCRYPT'06)*. Springer-Verlag, Berlin, Heidelberg, 339–358. https://doi.org/10.1007/11761679_21

[39] Yufeng Gu, Alireza Khadem, Sumanth Umesh, Ning Liang, Xavier Servot, Onur Mutlu, Ravi Iyer, and Reetuparna Das. 2025. *PIM Is All You Need: A CXL-Enabled GPU-Free System for Large Language Model Inference*. Association for Computing Machinery, New York, NY, USA, 862–881. https://doi.org/10.1145/3676641.3716267

[40] Mingxuan He, Choungki Song, Ilkon Kim, Chunseok Jeong, Seho Kim, Il Park, Mithuna Thottethodi, and T. N. Vijaykumar. 2020. Newton: A DRAM-maker's Accelerator-in-Memory (AiM) Architecture for Machine Learning. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 372–385. https://doi.org/10.1109/MICRO50266.2020.00040

[41] Yintao He, Haiyu Mao, Christina Giannoula, Mohammad Sadrosadati, Juan Gómez-Luna, Huawei Li, Xiaowei Li, Ying Wang, and Onur Mutlu. 2025. *PAPI: Exploiting Dynamic Parallelism in Large Language Model Decoding with a Processing-In-Memory-Enabled Computing System*. Association for Computing Machinery, New York, NY, USA, 766–782. https://doi.org/10.1145/3676641.3716009

[42] Guseul Heo, Sangyeop Lee, Jaehong Cho, Hyunmin Choi, Sanghyeon Lee, Hyungkyu Ham, Gwangsun Kim, Divya Mahajan, and Jongse Park. 2024. Neupims: Npu-pim heterogeneous acceleration for batched llm inferencing. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 722–737.

[43] Florian Hirner, Florian Krieger, Constantin Piber, and Sujoy {Sinha Roy}. 2025. Accelerating Hash-Based Polynomial Commitment Schemes with Linear Prover Time. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2025, 4 (5 Sept. 2025), 341–385. https://doi.org/10.46586/tches.v2025.i4.341-385

[44] Ingonyama. 2024. ICICLE: A GPU Library for Zero-Knowledge Proof Systems. https://github.com/ingonyama-zk/icicle Version 4.0.

[45] Heike Jagode, Anthony Danalis, Giuseppe Congiu, Daniel Barry, Anthony Castaldo, and Jack Dongarra. 2025. Advancements of PAPI for the exascale generation. *The International Journal of High Performance Computing Applications* 39, 2 (2025), 251–268. https://doi.org/10.1177/10943420241303884 arXiv:https://doi.org/10.1177/10943420241303884

[46] Je-Woo Jang, Junyong Oh, Youngbae Kong, Jae-Youn Hong, Sung-Hyuk Cho, Jeongyeol Lee, Hoeseok Yang, and Joon-Sung Yang. 2025. Accelerating Retrieval Augmented Language Model via PIM and PNM Integration. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture (MICRO '25)*. Association for Computing Machinery, New York, NY, USA, 246–262. https://doi.org/10.1145/3725843.3756020

[47] Taeyang Jeong and Eui-Young Chung. 2024. PipePIM: Maximizing Computing Unit Utilization in ML-Oriented Digital PIM by Pipelining and Dual Buffering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 12 (2024), 4585–4598. https://doi.org/10.1109/TCAD.2024.3410842

[48] Zhuoran Ji, Zhiyuan Zhang, Jiming Xu, and Lei Ju. 2024. Accelerating Multi-Scalar Multiplication for Efficient Zero Knowledge Proofs with Multi-GPU Systems *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 57–70. https://doi.org/10.1145/3620666.3651364

[49] Zhuoran Ji, Jianyu Zhao, Peimin Gao, Xiangkai Yin, and Lei Ju. 2025. Accelerating Number Theoretic Transform with Multi-GPU Systems for Efficient Zero Knowledge Proof. 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, 1–14. https://doi.org/10.1145/3669940.3707241

[50] Hai Jin, Dan Chen, Long Zheng, Yu Huang, Pengcheng Yao, Jin Zhao, Xiaofei Liao, and Wenbin Jiang. 2023. Accelerating graph convolutional networks through a pim-accelerated approach. *IEEE Trans. Comput.* 72, 9 (2023), 2628–2640.

[51] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, and Keith Kim. 2017. Hbm (high bandwidth memory) dram technology and architecture. In *2017 IEEE International Memory Workshop (IMW)*. IEEE, 1–4.

[52] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, and Keith Kim. 2017. HBM (High Bandwidth Memory) DRAM Technology and Architecture. In *2017 IEEE International Memory Workshop (IMW)*. 1–4. https://doi.org/10.1109/IMW.2017.7939084

[53] Mayank Kabra, Rakesh Nadig, Harshita Gupta, Rahul Bera, Manos Frouzakis, Vamanan Arulchelvan, Yu Liang, Haiyu Mao, Mohammad Sadrosadati, and Onur Mutlu. 2025. *CIPHERMATCH: Accelerating Homomorphic Encryption-Based String Matching via Memory-Efficient Data Packing and In-Flash Processing*. Association for Computing Machinery, New York, NY, USA, 111–130. https://doi.org/10.1145/3676641.3716251

[54] Hongju Kal, Chanyoung Yoo, and Won Woo Ro. 2023. AESPA: Asynchronous Execution Scheme to Exploit Bank-Level Parallelism of Processing-in-Memory. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture* (Toronto, ON, Canada) *(MICRO '23)*. Association for Computing Machinery, New York, NY, USA, 815–827. https://doi.org/10.1145/3613424.3614314

[55] Taewoon Kang, Geonwoo Choi, Taeweon Suh, and Gunjae Koo. 2025. SparsePIM: An Efficient HBM-Based PIM Architecture for Sparse Matrix-Vector Multiplications. In *Proceedings of the 39th ACM International Conference on Supercomputing (ICS '25)*. Association for Computing Machinery, New York, NY, USA, 495–512. https://doi.org/10.1145/3721145.3735111

[56] Jingyu Ke, Boxuan Liang, and Guoqiang Li. 2025. Consistency Verification for Zero-Knowledge Virtual Machine on Circuit-Irrelevant Representation. Cryptology ePrint Archive, Paper 2025/2204. https://eprint.iacr.org/2025/2204

[57] Liu Ke, Udit Gupta, Benjamin Youngjae Cho, David Brooks, Vikas Chandra, Utku Diril, Amin Firoozshahian, Kim Hazelwood, Bill Jia, Hsien-Hsin S. Lee, Meng Li, Bert Maher, Dheevatsa Mudigere, Maxim Naumov, Martin Schatz, Mikhail Smelyanskiy, Xiaodong Wang, Brandon Reagen, Carole-Jean Wu, Mark Hempstead, and Xuan Zhang. 2020. RecNMP: Accelerating Personalized Recommendation with Near-Memory Processing. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 790–803. https://doi.org/10.1109/ISCA45697.2020.00070

[58] Hyunwoong Kim, Jongcheol Park, Sanguk Lee, Jongwook Kim, and Seungyoung Ahn. 2023. Signal Integrity Analysis of Through-Silicon-Via (TSV) With Passive Equalizer to Separate Return Path and Mitigate the Inter-Symbol Interference (ISI) for Next Generation High Bandwidth Memory. *IEEE Transactions on Components, Packaging and Manufacturing Technology* PP (12 2023), 1–1. https://doi.org/10.1109/TCPMT.2023.3334789

[59] Jongmin Kim, Sungmin Yun, Hyesung Ji, Wonseok Choi, Sangpyo Kim, and Jung Ho Ahn. 2025. Anaheim: Architecture and Algorithms for Processing Fully Homomorphic Encryption in Memory. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1158–1173. https://doi.org/10.1109/HPCA61900.2025.00089

[60] Jin Hyun Kim, Shin-haeng Kang, Sukhan Lee, Hyeonsu Kim, Woongjae Song, Yuhwan Ro, Seungwon Lee, David Wang, Hyunsung Shin, Bengseng Phuah, Jihyun Choi, Jinin So, YeonGon Cho, JoonHo Song, Jangseok Choi, Jeonghyeon Cho, Kyomin Sohn, Youngsoo Sohn, Kwangil Park, and Nam Sung Kim. 2021. Aquabolt-XL: Samsung HBM2-PIM with in-memory processing for ML accelerators and beyond. In *2021 IEEE Hot Chips 33 Symposium (HCS)*. 1–26. https://doi.org/10.1109/HCS52781.2021.9567191

[61] Jung-Sik Kim, Chi Sung Oh, Hocheol Lee, Donghyuk Lee, Hyong-Ryol Hwang, Sooman Hwang, Byongwook Na, Joungwook Moon, Jin-Guk Kim, Hanna Park, Jang-Woo Ryu, Kiwon Park, Sang-Kyu Kang, So-Young Kim, Hoyoung Kim, Jong-Min Bang, Hyunyoon Cho, Minsoo Jang, Cheolmin Han, Jung-Bae Lee, Kyehyun Kyung, Joo-Sun Choi, and Young-Hyun Jun. 2011. A 1.2V 12.8GB/s 2Gb mobile Wide-I/O DRAM with 4×128 I/Os using TSV-based stacking. In *2011 IEEE International Solid-State Circuits Conference*. 496–498. https://doi.org/10.1109/ISSCC.2011.5746413

[62] Wonung Kim, Yubin Lee, Yoonsung Kim, Jinwoo Hwang, Seongryong Oh, Jiyong Jung, Aziz Huseynov, Woong Gyu Park, Chang Hyun Park, Divya Mahajan, and Jongse Park. 2025. Pimba: A Processing-in-Memory Acceleration for Post-Transformer Large Language Model Serving. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture (MICRO '25)*. Association for Computing Machinery, New York, NY, USA, 292–307. https://doi.org/10.1145/3725843.3756121

[63] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. 2016. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. In *2016 IEEE Symposium on Security and Privacy (SP)*. 839–858. https://doi.org/10.1109/SP.2016.55

[64] Daehan Kwon, Seongju Lee, Kyuyoung Kim, Sanghoon Oh, Joonhong Park, Gi-Moon Hong, Dongyoon Ka, Kyudong Hwang, Jeongje Park, Kyeongpil Kang, Jungyeon Kim, Junyeol Jeon, Nahsung Kim, Yongkee Kwon, Vladimir Kornijcuk, Woojae Shin, Jongsoon Won, Minkyu Lee, Hyunha Joo, Haerang Choi, Guhyun Kim, Byeongju An, Jaewook Lee, Donguc Ko, Younggun Jun, Ilwoong Kim, Choungki Song, Ilkon Kim, Chanwook Park, Seho Kim, Chunseok Jeong, Euicheol Lim, Dongkyun Kim, Jieun Jang, Il Park, Junhyun Chun, and Joohwan Cho. 2023. A 1ynm 1.25V 8Gb 16Gb/s/Pin GDDR6-Based Accelerator-in-Memory Supporting 1TFLOPS MAC Operation and Various Activation Functions for Deep Learning Application. *IEEE Journal of Solid-State Circuits* 58, 1 (2023), 291–302. https://doi.org/10.1109/JSSC.2022.3200718

[65] Yongkee Kwon, Kornijcuk Vladimir, Nahsung Kim, Woojae Shin, Jongsoon Won, Minkyu Lee, Hyunha Joo, Haerang Choi, Guhyun Kim, Byeongju An, Jeongbin Kim, Jaewook Lee, Ilkon Kim, Jaehan Park, Chanwook Park, Yosub Song, Byeongsu Yang, Hyungdeok Lee, Seho Kim, Daehan Kwon, Seongju Lee, Kyuyoung Kim, Sanghoon Oh, Joonhong Park, Gimoon Hong, Dongyoon Ka, Kyudong Hwang, Jeongje Park, Kyeongpil Kang, Jungyeon Kim, Junyeol Jeon, Myeongjun Lee, Minyoung Shin, Minhwan Shin, Jaekyung Cha, Changson Jung, Kijoon Chang, Chunseok Jeong, Euicheol Lim, Il Park, Junhyun Chun, and Sk Hynix. 2022. System Architecture and Software Stack for GDDR6-AiM. In *2022 IEEE Hot Chips 34 Symposium (HCS)*. 1–25. https://doi.org/10.1109/HCS55958.2022.9895629

[66] Young-Cheon Kwon, Suk Han Lee, Jaehoon Lee, Sang-Hyuk Kwon, Je Min Ryu, Jong-Pil Son, O Seongil, Hak-Soo Yu, Haesuk Lee, Soo Young Kim, Youngmin Cho, Jin Guk Kim, Jongyoon Choi, Hyun-Sung Shin, Jin Kim, BengSeng Phuah, HyoungMin Kim, Myeong Jun Song, Ahn Choi, Daeho Kim, SooYoung Kim, Eun-Bong Kim, David Wang, Shinhaeng Kang, Yuhwan Ro, Seungwoo Seo, JoonHo Song, Jaeyoun Youn, Kyomin Sohn, and Nam Sung Kim. 2021. 25.4 A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 64. 350–352. https://doi.org/10.1109/ISSCC42613.2021.9365862

[67] Dong Uk Lee, Ho Sung Cho, Jihwan Kim, Young Jun Ku, Sangmuk Oh, Chul Dae Kim, Hyun Woo Kim, Woo Young Lee, Tae Kyun Kim, Tae Sik Yun, Min Jeong Kim, SeungGyeon Lim, Seong Hee Lee, Byung Kuk Yun, Jun Il Moon, Ji Hwan Park, Seokwoo Choi, Young Jun Park, Chang Kwon Lee, Chunseok Jeong, Jae-Seung Lee, Sang Hun Lee, Woo Sung We, Jong Chan Yun, Doobock Lee, Junghyun Shin, Seungchan Kim, Junghwan Lee, Jiho Choi, Yucheon Ju, Myeong-Jae Park, Kang Seol Lee, Youngdo Hur, Daeyong Shim, Sangkwon Lee, Junhyun Chun, and Kyo-Won Jin. 2020. 22.3 A 128Gb 8-High 512GB/s HBM2E DRAM with a Pseudo Quarter Bank Structure, Power Dispersion and an Instruction-Based At-Speed PMBIST. In *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*. 334–336. https://doi.org/10.1109/ISSCC19947.2020.9062977

[68] Dong Uk Lee, Kyung Whan Kim, Kwan Weon Kim, Hongjung Kim, Ju Young Kim, Young Jun Park, Jae Hwan Kim, Dae Suk Kim, Heat Bit Park, Jin Wook Shin, Jang Hwan Cho, Ki Hun Kwon, Min Jeong Kim, Jaejin Lee, Kun Woo Park, Byongtae Chung, and Sungjoo Hong. 2014. 25.2 A 1.2V 8Gb 8-channel 128GB/s high-bandwidth memory (HBM) stacked DRAM with effective microbump I/O test methods using 29nm process and TSV. In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. 432–433. https://doi.org/10.1109/ISSCC.2014.6757501

[69] Jinhyung Lee, Kyungjun Cho, Chang Kwon Lee, Yeonho Lee, Jae-Hyung Park, Su-Hyun Oh, Yucheon Ju, Chunseok Jeong, Ho Sung Cho, Jaeseung Lee, Tae-Sik Yun, Jin Hee Cho, Sangmuk Oh, Junil Moon, Young-Jun Park, Hong-Seok Choi, In-Keun Kim, Seung Min Yang, Sun-Yeol Kim, Jaemin Jang, Jinwook Kim, Seong-Hee Lee, Younghyun Jeon, Juhyung Park, Tae-Kyun Kim, Dongyoon Ka, Sanghoon Oh, Jinse Kim, Junyeol Jeon, Seonhong Kim, Kyeong Tae Kim, Taeho Kim, Hyeonjin Yang, Dongju Yang, Minseop Lee, Heewoong Song, Dongwook Jang, Junghyun Shin, Hyunsik Kim, Changki Baek, Hajun Jeong, Jongchan Yoon, Seung-Kyun Lim, Kyo Yun Lee, Young Jun Koo, Myeong-Jae Park, Joohwan Cho, and Jonghwan Kim. 2024. 13.4 A 48GB 16-High 1280GB/s HBM3E DRAM with All-Around Power TSV and a 6-Phase RDQS Scheme for TSV Area Optimization. In *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 67. 238–240. https://doi.org/10.1109/ISSCC49657.2024.10454440

[70] Jong Chern Lee, Jihwan Kim, Kyung Whan Kim, Young Jun Ku, Dae Suk Kim, Chunseok Jeong, Tae Sik Yun, Hongjung Kim, Ho Sung Cho, Yeon Ok Kim, Jae Hwan Kim, Jin Ho Kim, Sangmuk Oh, Hyun Sung Lee, Ki Hun Kwon, Dong Beom Lee, Young Jae Choi, Jeajin Lee, Hyeon Gon Kim, Jun Hyun Chun, Jonghoon Oh, and Seok Hee Lee. 2016. 18.3 A 1.2V 64Gb 8-channel 256GB/s HBM DRAM with peripheral-base-die architecture and small-swing technique on heavy load interface. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. 318–319. https://doi.org/10.1109/ISSCC.2016.7418035

[71] Jong Chern Lee, Jihwan Kim, Kyung Whan Kim, Young Jun Ku, Dae Suk Kim, Chunseok Jeong, Tae Sik Yun, Hongjung Kim, Ho Sung Cho, Yeon Ok Kim, Jae Hwan Kim, Jin Ho Kim, Sangmuk Oh, Hyun Sung Lee, Ki Hun Kwon, Dong Beom Lee, Young Jae Choi, Jeajin Lee, Hyeon Gon Kim, Jun Hyun Chun, Jonghoon Oh, and Seok Hee Lee. 2016. 18.3 A 1.2V 64Gb 8-channel 256GB/s HBM DRAM with peripheral-base-die architecture and small-swing technique on heavy load interface. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. 318–319. https://doi.org/10.1109/ISSCC.2016.7418035

[72] Sukhan Lee, Shin-haeng Kang, Jaehoon Lee, Hyeonsu Kim, Eojin Lee, Seungwoo Seo, Hosang Yoon, Seungwon Lee, Kyounghwan Lim, Hyunsung Shin, Jinhyun Kim, O Seongil, Anand Iyer, David Wang, Kyomin Sohn, and Nam Sung Kim. 2021. Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology : Industrial Product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 43–56. https://doi.org/10.1109/ISCA52012.2021.00013

[73] Seongju Lee, Kyuyoung Kim, Sanghoon Oh, Joonhong Park, Gimoon Hong, Dongyoon Ka, Kyudong Hwang, Jeongje Park, Kyeongpil Kang, Jungyeon Kim, Junyeol Jeon, Nahsung Kim, Yongkee Kwon, Kornijcuk Vladimir, Woojae Shin, Jongsoon Won, Minkyu Lee, Hyunha Joo, Haerang Choi, Jaewook Lee, Donguc Ko, Younggun Jun, Keewon Cho, Ilwoong Kim, Choungki Song, Chunseok Jeong, Daehan Kwon, Jieun Jang, Il Park, Junhyun Chun, and Joohwan Cho. 2022. A 1ynm 1.25V 8Gb, 16Gb/s/pin GDDR6-based Accelerator-in-Memory supporting 1TFLOPS MAC Operation and Various Activation Functions for Deep-Learning Applications. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. 1–3. https://doi.org/10.1109/ISSCC42614.2022.9731711

[74] Cong Li, Zhe Zhou, Yang Wang, Fan Yang, Ting Cao, Mao Yang, Yun Liang, and Guangyu Sun. 2024. PIM-DL: Expanding the Applicability of Commodity DRAM-PIMs for Deep Learning via Algorithm-System Co-Optimization. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 879–896. https://doi.org/10.1145/3620665.3640376

[75] Dai Li, Akhil Pakala, and Kaiyuan Yang. 2022. MeNTT: A Compact and Efficient Processing-in-Memory Number Theoretic Transform (NTT) Accelerator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30, 5 (2022), 579–588. https://doi.org/10.1109/TVLSI.2022.3151321

[76] Muyang Li, Yueteng Yu, Bangyan Wang, Xiong Fan, and Shuwen Deng. 2025. ZKPoG: Accelerating WitGen-Incorporated End-to-End Zero-Knowledge Proof on GPU. Cryptology ePrint Archive, Paper 2025/765. https://eprint.iacr.org/2025/765

[77] Shang Li, Zhiyuan Yang, Dhiraj Reddy, Ankur Srivastava, and Bruce Jacob. 2020. DRAMsim3: A cycle-accurate, thermal-capable DRAM simulator. *IEEE Computer Architecture Letters* 19, 2 (2020), 106–109.

[78] Chenqi Lin, Kang Yang, Tianshi Xu, Ling Liang, Yufei Wang, Zhaohui Chen, Runsheng Wang, Mingyu Gao, and Meng Li. 2025. Ironman: Accelerating Oblivious Transfer Extension for Privacy-Preserving AI with Near-Memory Processing. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture (MICRO '25)*. Association for Computing Machinery, New York, NY, USA, 354–368. https://doi.org/10.1145/3725843.3756025

[79] Tianyi Liu, Tiancheng Xie, Jiaheng Zhang, Dawn Song, and Yupeng Zhang. 2024. Pianist: Scalable zkRollups via Fully Distributed Zero-Knowledge Proofs. In *Proceedings - 45th IEEE Symposium on Security and Privacy, SP 2024 (Proceedings - IEEE Symposium on Security and Privacy)*. Institute of Electrical and Electronics Engineers Inc., United States, 1777–1793. https://doi.org/10.1109/SP54263.2024.00035

[80] Tianyi Liu, Xiang Xie, and Yupeng Zhang. 2021. zkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 2968–2985. https://doi.org/10.1145/3460120.3485379

[81] Tao Lu, Yuxun Chen, Zonghui Wang, Xiaohang Wang, Wenzhi Chen, and Jiaheng Zhang. 2025. BatchZK: A Fully Pipelined GPU-Accelerated System for Batch Generation of Zero-Knowledge Proofs *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 100–115. https://doi.org/10.1145/3669940.3707270

[82] Tao Lu, Chengkun Wei, Ruijing Yu, Chaochao Chen, Wenjing Fang, Lei Wang, Zeke Wang, and Wenzhi Chen. 2022. cuZK: Accelerating Zero-Knowledge Proof with A Faster Parallel Multi-Scalar Multiplication Algorithm on GPUs. Cryptology ePrint Archive, Paper 2022/1321. https://eprint.iacr.org/2022/1321

[83] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. 1992. Algebraic methods for interactive proof systems. *J. ACM* 39, 4 (Oct. 1992), 859–868. https://doi.org/10.1145/146585.146605

[84] 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* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 340–353. https://doi.org/10.1145/3575693.3575711

[85] JEDEC Standard High Bandwidth Memory. 2015. Dram specification. *Standard JESD235A* (2015).

[86] JEDEC Standard High Bandwidth Memory. 2022. Dram specification. *Standard JESD238A* (2022).

[87] JEDEC Standard High Bandwidth Memory. 2025. Dram specification. *Standard JESD270-4A* (2025).

[88] Lifeng Nai, Ramyad Hadidi, Jaewoong Sim, Hyojong Kim, Pranith Kumar, and Hyesoon Kim. 2017. Graphpim: Enabling instruction-level pim offloading in graph computing frameworks. In *2017 IEEE International symposium on high performance computer architecture (HPCA)*. IEEE, 457–468.

[89] Kevin Nam, Heonhui Jung, Hyunyoung Oh, and Yunheung Paek. 2025. *Affinity-based Optimizations for TFHE on Processing-in-DRAM*. Association for Computing Machinery, New York, NY, USA, 16–31. https://doi.org/10.1145/3676641.3716246

[90] Hamid Nejatollahi, Saransh Gupta, Mohsen Imani, Tajana Simunic Rosing, Rosario Cammarota, and Nikil Dutt. 2020. CryptoPIM: In-memory Acceleration for Lattice-based Cryptographic Hardware. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC18072.2020.9218730

[91] Dimin Niu, Shuangchen Li, Yuhao Wang, Wei Han, Zhe Zhang, Yijin Guan, Tianchan Guan, Fei Sun, Fei Xue, Lide Duan, Yuanwei Fang, Hongzhong Zheng, Xiping Jiang, Song Wang, Fengguo Zuo, Yubing Wang, Bing Yu, Qiwei Ren, and Yuan Xie. 2022. 184QPS/W 64Mb/mm23D Logic-to-DRAM Hybrid Bonding with Process-Near-Memory Engine for Recommendation System. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. 1–3. https://doi.org/10.1109/ISSCC42614.2022.9731694

[92] NVIDIA Corporation. 2025. NVIDIA Nsight Compute 2025.1. https://developer.nvidia.com/nsight-compute

[93] NVIDIA Corporation. 2025. *NVML API Reference Guide*. https://docs.nvidia.com/deploy/nvml-api/index.html Version R590.

[94] Chi-Sung Oh, Ki Chul Chun, Young-Yong Byun, Yong-Ki Kim, So-Young Kim, Yesin Ryu, Jaewon Park, Sinho Kim, Sanguhn Cha, Donghak Shin, Jungyu Lee, Jong-Pil Son, Byung-Kyu Ho, Seong-Jin Cho, Beomyong Kil, Sungoh Ahn, Baekmin Lim, Yongsik Park, Kijun Lee, Myung-Kyu Lee, Seungduk Baek, Junyong Noh, Jae-Wook Lee, Seungseob Lee, Sooyoung Kim, Botak Lim, Seouk-Kyu Choi, Jin-Guk Kim, Hye-In Choi, Hyuk-Jun Kwon, Jun Jin Kong, Kyomin Sohn, Nam Sung Kim, Kwang-Il Park, and Jung-Bae Lee. 2020. 22.1 A 1.1V 16GB 640GB/s HBM2E DRAM with a Data-Bus Window-Extension Technique and a Synergetic On-Die ECC Scheme. In *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*. 330–332. https://doi.org/10.1109/ISSCC19947.2020.9063110

[95] Cristobal Ortega, Yann Falevoz, and Renaud Ayrignac. 2024. PIM-AI: A Novel Architecture for High-Efficiency LLM Inference. *arXiv preprint arXiv:2411.17309* (2024).

[96] Jaehyun Park, Jaewan Choi, Kwanhee Kyung, Michael Jaemin Kim, Yongsuk Kwon, Nam Sung Kim, and Jung Ho Ahn. 2024. AttAcc! Unleashing the Power of PIM for Batched Transformer-based Generative Model Inference. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 103–119. https://doi.org/10.1145/3620665.3640422

[97] Jaewoo Park, Sugil Lee, and Jongeun Lee. 2023. NTT-PIM: Row-Centric Architecture and Mapping for Efficient Number-Theoretic Transform on PIM. In *Proc. Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC56929.2023.10247747

[98] Myeong-Jae Park, Jinhyung Lee, Kyungjun Cho, Jihwan Park, Junil Moon, Sung-Hak Lee, Tae-Kyun Kim, Sanghoon Oh, Seokwoo Choi, Yongsuk Choi, Ho Sung Cho, Taesik Yun, Young Jun Koo, Jae-Seung Lee, Byung-Kuk Yoon, Young-Jun Park, Sangmuk Oh, Chang Kwon Lee, Seong-Hee Lee, Hyun-Woo Kim, Yucheon Ju, Seung-Kyun Lim, Kyo Yun Lee, Sang-Hoon Lee, Woo Sung We, Seungchan Kim, Seung Min Yang, Keonho Lee, In-Keun Kim, Younghyun Jeon, Jae-Hyung Park, Jong Chan Yun, Seonyeol Kim, Dong-Yeol Lee, Su-Hyun Oh, Jung-Hyun Shin, Yeonho Lee, Jieun Jang, and Joohwan Cho. 2023. A 192-Gb 12-High 896-GB/s HBM3 DRAM With a TSV Auto-Calibration Scheme and Machine-Learning-Based Layout Optimization. *IEEE Journal of Solid-State Circuits* 58, 1 (2023), 256–269. https://doi.org/10.1109/JSSC.2022.3193354

[99] Naebeom Park, Sungju Ryu, Jaeha Kung, and Jae-Joon Kim. 2021. High-throughput Near-Memory Processing on CNNs with 3D HBM-like Memory. *ACM Trans. Des. Autom. Electron. Syst.* 26, 6, Article 48 (June 2021), 20 pages. https://doi.org/10.1145/3460971

[100] Seth H Pugsley, Jeffrey Jestes, Huihui Zhang, Rajeev Balasubramonian, Vijayalakshmi Srinivasan, Alper Buyuktosunoglu, Al Davis, and Feifei Li. 2014. NDC: Analyzing the impact of 3D-stacked memory+ logic devices on MapReduce workloads. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 190–200.

[101] Andy Ray, Benjamin Devlin, Fu Yong Quah, and Rahul Yesantharao. 2024. Hardcaml MSM: A High-Performance Split CPU-FPGA Multi-Scalar Multiplication Engine. In *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (Monterey, CA, USA) *(FPGA '24)*. Association for Computing Machinery, New York, NY, USA, 33–39. https://doi.org/10.1145/3626202.3637577

[102] Mohammadreza Saed, Prashant J. Nair, and Tor M. Aamodt. 2025. RayN: Ray Tracing Acceleration with Near-memory Computing. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture (MICRO '25)*. Association for Computing Machinery, New York, NY, USA, 277–291. https://doi.org/10.1145/3725843.3756067

[103] 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. https://doi.org/10.1109/MICRO61859.2024.00035

[104] Satyabrata Sarangi and Bevan Baas. 2021. DeepScaleTool: A Tool for the Accurate Estimation of Technology Scaling in the Deep-Submicron Era. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–5. https://doi.org/10.1109/ISCAS51556.2021.9401196

[105] Alessandra Scafuro, Luisa Siniscalchi, and Ivan Visconti. 2019. Publicly Verifiable Proofs from Blockchains. In *Public-Key Cryptography – PKC 2019: 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part I* (Beijing, China). Springer-Verlag, Berlin, Heidelberg, 374–401. https://doi.org/10.1007/978-3-030-17253-4_13

[106] Minseok Seo, Xuan Truong Nguyen, Seok Joong Hwang, Yongkee Kwon, Guhyun Kim, Chanwook Park, Ilkon Kim, Jaehan Park, Jeongbin Kim, Woojae Shin, Jongsoon Won, Haerang Choi, Kyuyoung Kim, Daehan Kwon, Chunseok

Jeong, Sangheon Lee, Yongseok Choi, Wooseok Byun, Seungcheol Baek, Hyuk-Jae Lee, and John Kim. 2024. IANUS: Integrated Accelerator based on NPU-PIM Unified Memory System. 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, 545–560. https://doi.org/10.1145/3620666.3651324

[107] Srinath Setty. 2019. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. Cryptology ePrint Archive, Paper 2019/550. https://eprint.iacr.org/2019/550

[108] Srinath Setty, Justin Thaler, and Riad Wahby. 2023. Unlocking the lookup singularity with Lasso. Cryptology ePrint Archive, Paper 2023/1216. https://eprint.iacr.org/2023/1216

[109] Kyomin Sohn, Won-Joo Yun, Reum Oh, Chi-Sung Oh, Seong-Young Seo, Min-Sang Park, Dong-Hak Shin, Won-Chang Jung, Sang-Hoon Shin, Je-Min Ryu, Hye-Seung Yu, Jae-Hun Jung, Kyung-Woo Nam, Seouk-Kyu Choi, Jae-Wook Lee, Uksong Kang, Young-Soo Sohn, Jung-Hwan Choi, Chi-Wook Kim, Seong-Jin Jang, and Gyo-Young Jin. 2016. 18.2 A 1.2V 20nm 307GB/s HBM DRAM with at-speed wafer-level I/O test scheme and adaptive refresh considering temperature distribution. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. 316–317. https://doi.org/10.1109/ISSCC.2016.7418034

[110] Boyu Tian, Qihang Chen, and Mingyu Gao. 2023. Abndp: Co-optimizing data access and load balance in near-data processing. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 3–17.

[111] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. 2018. Doubly-Efficient zkSNARKs Without Trusted Setup. In *2018 IEEE Symposium on Security and Privacy (SP)*. 926–943. https://doi.org/10.1109/SP.2018.00060

[112] 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. https://doi.org/10.1145/3669940.3707228

[113] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. 2018. DIZK: A Distributed Zero Knowledge Proof System. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 675–692. https://www.usenix.org/conference/usenixsecurity18/presentation/wu

[114] Yu Xia, Xiangyao Yu, Matthew Butrovich, Andrew Pavlo, and Srinivas Devadas. 2022. Litmus: Towards a Practical Database Management System with Verifiable ACID Properties and Transaction Correctness. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1478–1492. https://doi.org/10.1145/3514221.3517851

[115] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. 2022. zkBridge: Trustless Cross-chain Bridges Made Practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 3003–3017. https://doi.org/10.1145/3548606.3560652

[116] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. 2019. Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation. In *Advances in Cryptology – CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III* (Santa Barbara, CA, USA). Springer-Verlag, Berlin, Heidelberg, 733–764. https://doi.org/10.1007/978-3-030-26954-8_24

[117] Zhibo Xing, Zijian Zhang, Ziang Zhang, Zhen Li, Meng Li, Jiamou Liu, Zongyang Zhang, Yi Zhao, Qi Sun, Liehuang Zhu, and Giovanni Russello. 2026. Zero-Knowledge Proof-Based Verifiable Decentralized Machine Learning in Communication Network: A Comprehensive Survey. *IEEE Communications Surveys & Tutorials* 28 (2026), 985–1024. https://doi.org/10.1109/COMST.2025.3561657

[118] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. 2021. QuickSilver: Efficient and Affordable Zero-Knowledge Proofs for Circuits and Polynomials over Any Field. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 2986–3001. https://doi.org/10.1145/3460120.3484556

[119] Weidong Yang, Yuqing Yang, Shuya Ji, Jianfei Jiang, Naifeng Jing, Qin Wang, Zhigang Mao, and Weiguang Sheng. 2024. RecPIM: Efficient In-Memory Processing for Personalized Recommendation Inference Using Near-Bank Architecture. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2024).

[120] Yinghao Yang, Hang Lu, and Xiaowei Li. 2023. Poseidon-NDP: Practical Fully Homomorphic Encryption Accelerator Based on Near Data Processing Architecture. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 12 (2023), 4749–4762. https://doi.org/10.1109/TCAD.2023.3292211

[121] 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. https://doi.org/10.1109/HPCA61900.2025.00020

[122] Byungkuk Yoon, Sanghyeok Han, Gyeonghwan Park, and Jae-Joon Kim. 2025. SplitSync: Bank Group-Level Split-Synchronization for High-Performance DRAM PIM. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. 1–7. https://doi.org/10.1109/DAC63849.2025.11132821

[123] Douglas Yu. 2021. TSMC Packaging Technologies for Chiplets and 3D. In *Hot Chips 33 Symposium (HCS)*. https://hc33.hotchips.org/assets/program/tutorials/2021%20HotChips%20TSMC%20Packaging%20Technologies%20for%20Chiplets%20and%203D_0819%20publish_public.pdf

[124] Zhiheng Yue, Yang Wang, Chao Li, Shaojun Wei, Yang Hu, and Shouyi Yin. 2025. 3D-PATH: A Hierarchy LUT Processing-in-memory Accelerator with Thermal-aware Hybrid Bonding Integration. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture (MICRO '25)*. Association for Computing Machinery, New York, NY, USA, 78–93. https://doi.org/10.1145/3725843.3756087

[125] Dongping Zhang, Nuwan Jayasena, Alexander Lyashevsky, Joseph L Greathouse, Lifan Xu, and Michael Ignatowski. 2014. TOP-PIM: Throughput-oriented programmable processing in memory. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*. 85–98.

[126] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. 2019. Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof. Cryptology ePrint Archive, Paper 2019/1482. https://eprint.iacr.org/2019/1482

[127] Mingxing Zhang, Youwei Zhuo, Chao Wang, Mingyu Gao, Yongwei Wu, Kang Chen, Christos Kozyrakis, and Xuehai Qian. 2018. GraphP: Reducing communication for PIM-based graph processing with efficient data partition. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 544–557.

[128] Yuncong Zhang, Shi-Feng Sun, Ren Zhang, and Dawu Gu. 2023. Polynomial IOPs for Memory Consistency Checks in Zero-Knowledge Virtual Machines. In *Advances in Cryptology – ASIACRYPT 2023*, Jian Guo and Ron Steinfeld (Eds.). Springer Nature Singapore, Singapore, 111–141.

[129] Ye Zhang, Shuo Wang, Xian Zhang, Jiangbin Dong, Xingzhong Mao, Fan Long, Cong Wang, Dong Zhou, Mingyu Gao, and Guangyu Sun. 2021. PipeZK: Accelerating Zero-Knowledge Proof with a Pipelined Architecture. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 416–428. https://doi.org/10.1109/ISCA52012.2021.00040

[130] Baoze Zhao, Wenjin Huang, Tianrui Li, and Yihua Huang. 2023. BSTMSM: A High-Performance FPGA-based Multi-Scalar Multiplication Hardware Accelerator. In *2023 International Conference on Field Programmable Technology (ICFPT)*. 35–43. https://doi.org/10.1109/ICFPT59805.2023.00009

[131] Minxuan Zhou, Yujin Nam, Pranav Gangwar, Weihong Xu, Arpan Dutta, Chris Wilkerson, Rosario Cammarota, Saransh Gupta, and Tajana Rosing. 2025. FHEmem: A Processing In-Memory Accelerator for Fully Homomorphic Encryption. *IEEE Transactions on Emerging Topics in Computing* 13, 4 (2025), 1367–1382. https://doi.org/10.1109/TETC.2025.3528862

[132] Minxuan Zhou, Weihong Xu, Jaeyoung Kang, and Tajana Rosing. 2022. TransPIM: A Memory-based Acceleration via Software-Hardware Co-Design for Transformer. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 1071–1085. https://doi.org/10.1109/HPCA53966.2022.00082

[133] Qiuling Zhu, Berkin Akin, H Ekin Sumbul, Fazle Sadi, James C Hoe, Larry Pileggi, and Franz Franchetti. 2013. A 3D-stacked logic-in-memory accelerator for application-specific data intensive computing. In *2013 IEEE international 3D systems integration conference (3DIC)*. IEEE, 1–7.

[134] Qiuling Zhu, Tobias Graf, H Ekin Sumbul, Larry Pileggi, and Franz Franchetti. 2013. Accelerating sparse matrix-matrix multiplication with 3D-stacked logic-in-memory hardware. In *2013 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–6.

[135] Youwei Zhuo, Chao Wang, Mingxing Zhang, Rui Wang, Dimin Niu, Yanzhi Wang, and Xuehai Qian. 2019. Graphq: Scalable pim-based graph processing. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 712–725.