Chiplet-Based Techniques for Scalable and Memory-Aware Multi-Scalar Multiplication

Florian Hirner
Graz University of Technology, Austria
florian.hirner@iaik.tugraz.at

Florian Krieger
Graz University of Technology, Austria
florian.krieger@iaik.tugraz.at

Sujoy Sinha Roy
Graz University of Technology, Austria
sujoy.sinharoy@iaik.tugraz.at

***Abstract*—This paper presents a high-performance architecture for accelerating Multi-Scalar Multiplication (MSM) on ASIC platforms, targeting cryptographic applications with high throughput demands. Unlike prior MSM accelerators that focus solely on efficient processing elements (PEs), our chiplet-based design optimally balances area, power, and computational throughput. We identify a mixed window configuration of 12- and 13-bit windows that enables an efficient multi-PE integration of 10 PEs per chiplet. Our single-PE design achieves a 1.37x speedup and 1.3x area reduction over prior works, while the multi-PE chiplet design improves the area-time product by 2.2x, offering scalability, lower production costs, and higher manufacturing yields.**

***Index Terms*—Multi-Scalar Multiplication, Zero-knowledge proofs, Hardware Acceleration, Scalable Chiplet Architecture, Parallel Computing**

I. Introduction

Multi-Scalar Multiplication (MSM) is a computationally intensive operation in cryptographic protocols, particularly in pairing-based zero-knowledge proofs (ZKPs) [1]. As shown in Figure 1, MSM accounts for over 70% of proof generation time [2]. Its computational dominance increases with the number of points N, as seen in systems like Groth16 [3], PlonK-KZG [4], and Marlin-KZG [5], which process tens to hundreds of gigabytes of data and take minutes for 2^{25} to 2^{28} points. Hardware acceleration of MSM is crucial for improving the time and energy of proof generation.

CycloneMSM [6] and Hardcaml [7] are high-performance FPGA-based MSM accelerators from the 2023 Z-Prize [8] challenge. These designs leverage Pippenger's bucket method [9] to accelerate MSM by decomposing [10] it into three key steps: bucket classification, bucket aggregation, and result aggregation. The two accelerators optimize the computationally dominant bucket classification step and use a single fully pipelined point adder unit. However, due to the large size of fully pipelined point adder, these accelerators cannot instantiate more than one processing element (PE) even in high-end FPGAs. ASIC architectures such as PipeZK [11], Gypsophila [12], and PriorMSM [13] primarily employ single processing element (PE) configurations with optimized memory access strategies to mitigate bandwidth pressure from bucket collisions during classification. Among these,

This project was funded in part by the State Government of Styria, Austria – Department Zukunftsfonds Steiermark.

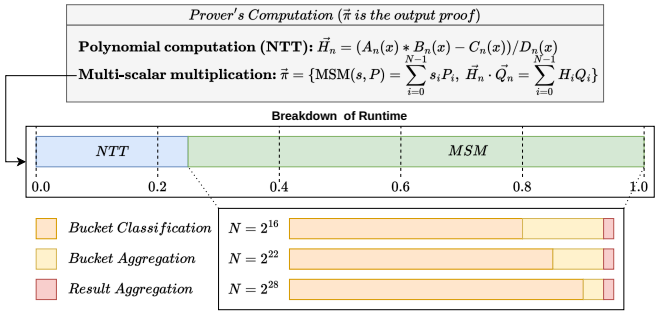

Fig. 1. Overview of workload distribution for NTT, MSM, and Bucketing operations involved in proof generation of paring-based ZKPs.

only Gypsophila [12] explores a multi-PE design where each PE independently computes separate MSMs.

Notably, no prior work has investigated a collaborative multi-PE MSM design where all PEs work together on a single MSM computation. Such a design could significantly reduce latency and enhance throughput.

Motivation for chiplet-based MSM design approach: Instantiating multiple PEs for MSM increases the overall area of the ASIC design, which can escalate production costs and negatively impact yield due to the complexities associated with fabricating large chips. For example, as shown in Table I, a monolithic ASIC accelerator with 16-bit window size and 16 PEs would require approximately 269mm^2 of die area which is large. In modern semiconductor engineering, multi-chiplet System-in-Package (SiP) provides an effective solution to address cost and yield challenges of large monolithic chips. Decomposing the design into smaller and modular chiplets not only improves scalability and manufacturing yield but also reduces production costs [14]. Recent works – such as CiFHer [15], REED [16] and Chiplever [17] – have explored the adoption of chiplet-based architectures for Fully Homomorphic Encryption (FHE), which like MSM is also computation and data intensive. This motivates us to explore the research question: *How can MSM be implemented as a multi-chiplet SiP to achieve high performance?*
Our contributions are summarized as follows:

- **Memory aware PE Design:** We analyze the impact of memory footprint and latency on PEs and propose an optimized design that balances area and computational throughput for MSM workloads.

- **Mixed window sizes:** We introduce mixed window size configurations for multi-PE setups, demonstrating how they optimize memory usage and workload distribution while minimizing latency.
- **Inter-chiplet communication:** We propose diverse workload distribution and interconnect strategies that minimize inter-chiplet and off-chip memory communications. We use these distribution strategies (vertical, horizontal or full unrolling) to propose three different chiplet-based architectures.
- **Comprehensive evaluation:** We evaluate our design across varying configurations, showing up to 1.14× and 2.2× improvements in throughput and area efficiency, respectively, compared to state-of-the-art monolithic designs. In addition, our chiplet-based design enhances the scalability and production yield for ASIC platforms.

II. Background

This section outlines the fundamental aspects of MSM, including its mathematical foundation in elliptic curve cryptography and the Pippenger algorithm used to optimize MSM computations.

A. Elliptic Curves

An elliptic curve E over a finite field $\mathbb{F}_q$ is defined as the set of points $(x, y) \in \mathbb{F}_q^2$ that satisfy the Weierstrass equation $y^2 = x^3 + ax + b$. For appropriately chosen curve parameters a and b, the points on E form an Abelian group $(E, +)$, where the group operation, known as *point addition*, combines two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ to produce a third point $P_3 = (x_3, y_3)$ on E. Point addition is governed by specific rules of the elliptic curve and involves field arithmetic operations in $\mathbb{F}_q$. The group's identity element is the point at infinity, denoted by $\mathcal{O}$. For a positive integer scalar s, the scalar multiplication of a point $P \in E$ is defined as $s \cdot P = \sum_{k=1}^{s} P$, which corresponds to adding P to itself s times.

Elliptic curve arithmetic is the backbone of MSM. The choice of curve forms and coordinate systems influence the cost of elliptic curve arithmetic. Adding two points on the Weierstrass curve involves several modular additions, modular multiplications, and in some cases, very expensive modular inversions in the field $\mathbb{F}_q$. These finite field operations are computationally intensive, as the prime q is usually 384 bits or similar in pairing-based ZKPs, necessitating efficient implementation techniques. The choice of coordinate system to represent points on the curve influences the cost of point addition. For example, projective coordinates eliminate the need for modular inversion by introducing additional modular multiplications. Extended projective coordinates (X, Y, Z, T) are particularly efficient for Twisted Edwards curves, a specialized form of elliptic curves, as they minimize the number of modular multiplications in point addition [18].

B. Multi-Scalar Multiplications

Multi-Scalar Multiplication (MSM) is a fundamental operation in elliptic curve cryptography and zero-knowledge proofs, where the goal is to compute a weighted sum of elliptic curve points. Given a set of scalars $s = (s_0, s_1, \ldots, s_{N-1})$ with 253-bit $s_i \in \mathbb{F}_p \subseteq \mathbb{N}$ and points $P = (P_0, P_1, \ldots, P_{N-1})$ on an elliptic curve E, MSM is expressed as:

$$\text{MSM}(s, P) = s_0 \cdot P_0 + s_1 \cdot P_1 + \cdots + s_{N-1} \cdot P_{N-1}. \quad (1)$$

Since N easily reaches 2^{26} in zero-knowledge proofs, the computational cost of MSM is substantial, driving the need for efficient algorithms and hardware acceleration. Given the computational intensity of MSM, efficient algorithms like the *Pippenger algorithm* [9] are crucial for minimizing the number of scalar multiplications and achieving high-performance hardware implementations.

Algorithm 1 Pippenger Algorithm - Bucket Method [12]

function BUCKETCLASSIFICATION(s,P)
  Set $B_{j,k} \leftarrow \mathcal{O}\ \forall j, k$
  for $j = 0$ **to** $m - 1$ **do**
    for $i = 0$ **to** $N - 1$ **do**
      $k \leftarrow s_{i,j}$
      $B_{j,k} \leftarrow B_{j,k} + P_i$
    end for
  end for
  return $B_{j,k}$
end function
function BUCKETAGGREGATION($B_{j,k}$)
  $S_j \leftarrow \mathcal{O}\ \forall j$
  for $j = 0$ **to** $m - 1$ **do**
    for $k = 0$ **to** $2^w - 1$ **do**
      $S_j \leftarrow S_j + k \cdot B_{j,k}$
    end for
  end for
  return S_j
end function
function RESULTAGGREGATION(S_j)
  $R \leftarrow \mathcal{O}$
  for $j = m - 1$ **downto** 0 **do**
    $R \leftarrow 2^w \cdot R$
    $R \leftarrow R + S_j$
  end for
  return R ▷ $R = MSM(s, P)$
end function

C. Pippenger Algorithm

The Pippenger algorithm [9], widely employed in MSM, structures computations to maximize parallelism and reduce the overall number of scalar multiplications. Algorithm 1 shows Pippenger's method. This algorithm divides each 253-bit scalar s_i into w-bit wide windows $s_{i,j}$, such that $s_i = \sum_{j=0}^{m-1} s_{i,j} \cdot 2^{j \cdot w}$, where $m = \lceil \frac{253}{w} \rceil$. This enables independent processing of *smaller* scalar segments $s_{i,j}$.

The Pippenger algorithm executes three main stages: (1) Bucket Classification, (2) Bucket Aggregation, and (3) Result Aggregation, as shown in Algorithm 1. In the Bucket Classification, the value of windows $s_{i,j}$ classifies points into buckets. Specifically, each window value $s_{i,j}$ determines the bucket $B_{j,s_{i,j}}$ to which the point P_i belongs. This classification can be represented as: $B_{j,k} = \sum P_i \mid s_{i,j} = k$. Therein, $B_{j,k}$ is the bucket for the j-th window containing all points P_i

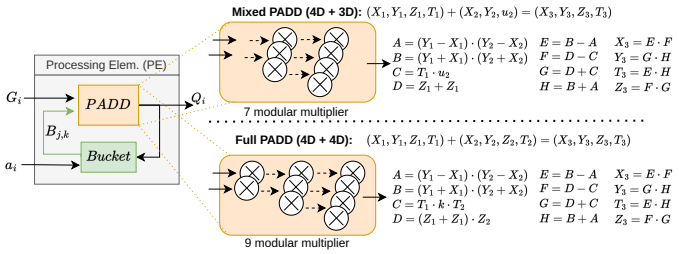

Fig. 2. Overview of mixed (top) and full (bottom) PADD modules.

with scalar values $s_{i,j} = k$. Aggregating points into buckets reduces the number of scalar multiplications needed, as each bucket effectively represents a partial sum within the MSM computation. Once bucket classification is complete, the algorithm computes the sum S_j over all buckets $B_{j,k}$ of window j. Formally, it computes $S_j = \sum_{k=0}^{2^w-1} k \cdot B_{j,k}$. In the final stage, it combines the results S_j from all windows by summing each contribution, weighted by the position of the window. The final MSM computation is: $\text{MSM}(s, P) = \sum_{j=0}^{m-1} S_j \cdot 2^{j \cdot w}$. Each window's sum S_j is scaled by a factor of $2^{j \cdot w}$ to account for the window's position, and the weighted sums are then combined to get the MSM result.

III. Proposed Design

This section details the design of our MSM accelerator, with a focus on how we optimize performance, scalability, and resource utilization. We begin by describing the architecture of the Processing Element (PE), including the trade-offs involved in selecting the appropriate window size. Next, we address the challenges in scaling MSM operations by distributing workloads across multiple PEs and efficiently managing memory bandwidth. Finally, we present our chiplet-based approach, discussing how we map operations to multiple chiplets and optimize the design for both DDR and high-bandwidth memory (HBM) configurations through horizontal and vertial unrolling techniques.

A. Processing Element

The fundamental building block of our MSM accelerator is an optimized Processing Element (PE) design tailored to handle point addition and doubling operations on elliptic curves. Each PE consists of two main subunits: a point adder (PADD) and bucket memory for storing results of bucket classification.

Figure 2 illustrates two different PE designs, where the top PE shows a mixed PADD unit and the bottom shows a full PADD unit. Previous works [6], [7], [19] on FPGA typically use mixed PADD units to reduce area requirements, as a mixed PADD unit requires only seven instead of nine modular multiplication units. Note that each modular multiplier operates on 377-bit operands via Barret reduction [20]. Within our PE design, we used the open-source modular multiplier of [6]. Although a mixed PADD reduces area, it restricts the PE's efficiency by requiring multiple passes through the unit to simulate a full PADD. This restricts the efficiency especially for bucket and result aggregation.

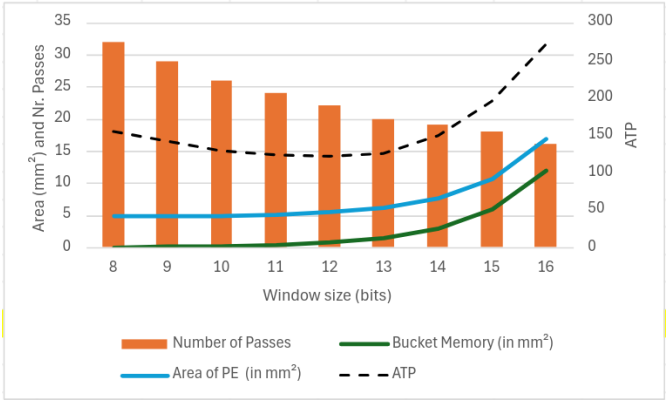

Fig. 3. Impact of window size on area consumption, number of passes, and area-time product.

In contrast to previous work we utilize a full PADD unit, this approach allows each PE to perform complete bucket classification and aggregation without intermediate results or multiple passes. This maintaining a smooth pipeline flow and achieving higher overall efficiency during bucket- and result aggregation. It is noteworthy that the usage of a full PADD consumes more resources compared to mixed PADD designs, however, it avoids the complexity of pass-by-pass accumulation during the aggregation steps.

1) Impact of Window Size on Bucket Memory and Latency for single PE case: To evaluate the impact of different window sizes, we first analyze a single PE performing the entire MSM computation. The window size directly influences both the PE's area requirements and the overall MSM latency. Larger window sizes reduce the number of computation passes by processing larger fractions of the scalar in each pass, thereby decreasing latency. However, this comes at the cost of exponentially increased bucket memory usage. Conversely, smaller window sizes keep memory usage manageable but require more computation passes, potentially increasing latency.

To determine the optimal configuration for our single PE design, we analyzed window sizes ranging from 8 to 16 bits. Figure 3 illustrates the relationships between the bucket memory area (in mm^2), total PE area (in mm^2), number of computation passes, and the Area-Time Product (ATP). These datasets highlight how window size impacts both performance and resource requirements.

Our analysis reveals that the bucket memory area doubles with each increment in window size, leading to exponential growth. Meanwhile, the reduction in computation passes provides a roughly linear performance gain. This mismatch results in exponential growth of the ATP (black dotted trace in Figure 3), with a minimum observed in the range of 11- to 13-bit windows. For window sizes up to 13 bits, the bucket memory has minimal impact on the total PE area, which remains dominated by the PADD module. Beyond 13-bit windows, however, the PE area increases significantly

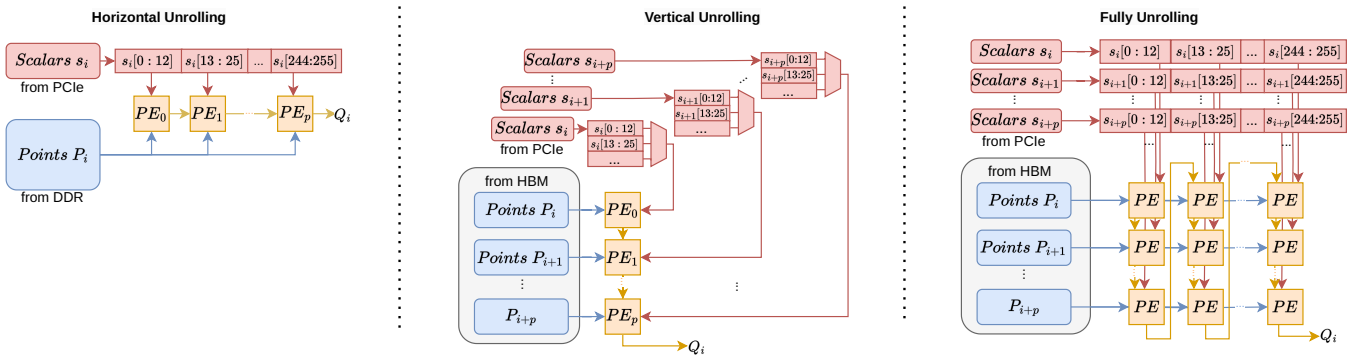

Fig. 4. Overview of different distribution techniques for vertical, horizontal and full unrolling.

TABLE I
COST OF MONOLITHIC DESIGN FOR VARIOUS WINDOW SIZES

Window (in bit)	8	9	10	11	12	13	14	15	16
Num. of PEs	32	29	26	24	22	20	19	18	16
Area (in mm2)	154	141	129	124	121	125	149	195	269
Power (in W)	122	111	99	91	84	76	72	68	61

due to the growing memory requirements. Based on these findings, we select a 13-bit window size as the optimal balance between performance and area efficiency for our single PE configuration.

2) Impact of Window Size on Fully Unrolled Architecture with multi-PE case: After discussing the single-PE case, we now focus on a fully unrolled PE array. In the fully unrolled configuration, the entire scalar multiplication $s_i \cdot P_i$ is computed in parallel across multiple PEs. Each PE operates on a specific window w_i of the scalar, where $s_i = \langle w_0 | w_1 | \dots | w_{c-1} \rangle$. The window size determines the granularity of these operations, directly influencing both the utilization of individual PEs and the overall design of the fully unrolled array.

Table I illustrates the trade-offs between area and power consumption for various window sizes ranging from 8 to 16 bits. The first row of Table I shows the number of PEs required to cover a 256-bit scalar. For instance, with an 8-bit window, 32 PEs are required, each with a smaller bucket memory. Conversely, larger windows reduce the number of PEs but require more memory per PE, as discussed earlier. Interestingly, the overall area consumption in the fully unrolled case is minimal for 11 to 13-bit windows, which is almost the same window size as found optimal for the single PE case. This again highlights the reasonability of our chosen window size of 13 bits.

In terms of power consumption, Table I shows a decline for larger window sizes. This trend arises because, as window sizes grow, the memory footprint increases while the number of PEs decreases. Since the modular multipliers in PEs consume more power than memory, larger windows lead to reduced overall power requirements.

For example, a 16-bit window configuration, as used in [6], [12], requires a total area of 269 mm^2 and 61 W of power when fully unrolled – when taking TSMC 28nm node. These resource requirements limit scalability due to increased area and power demands. By contrast, our analysis identifies an optimal window size range of 11 to 13 bits, which balances area and latency efficiency. This range allows the implementation of multiple PEs without incurring excessive area or power costs.

1) Bucket Aggregation and Final Sum: After completing bucket classification, the points in each bucket must be aggregated to compute the final result of the bucket. Traditional approaches rely on an iterative computation flow for bucket aggregation, where points are summed sequentially within each bucket via Double-and-Add similar to the fast exponentiation algorithm [21]. This method is highly sensitive to window size, as larger windows result in larger bucket sizes and therefore more aggregation steps. Consequently, the iterative nature of this approach introduces latency bottlenecks due to the pipelined nature of the PADD unit. To address this, our design overlaps bucket aggregation with classification, parallelizing these operations to improve throughput.

B. Chiplet-based Considerations

The previous sections identified 11 to 13 bits as the optimal window size range for single-PE and fully unrolled PE arrays. Monolithic MSM designs, however, face scalability challenges due to the exponential dependency of the die area on window size, as shown in Table I. Although smaller windows increase power consumption by requiring more PEs, larger windows reduce PEs while significantly increasing memory area. This exponential growth in area highlights the limitations of monolithic implementations as they lead to high production costs.

Distribution MSM Workloads Across PEs: We propose techniques to distribute MSM workloads across multiple PEs on separate chiplets, avoiding reliance on a single large die. These strategies reduce the total number of off-chip memory accesses, improving throughput and scalability. Figure 4 illustrates three different workload distributions for chiplet-based designs with two different memory configurations: (1) low-bandwidth DDR (Double Data Rate) and (2) high-bandwidth HBM (High-Bandwidth Memory).

(1) In low-bandwidth setups – shown on the left side of Figure 4 – horizontal unrolling minimizes redundant point loading from off-chip memory by processing all windows of the scalar in parallel. Each point is loaded only once, and computations for all scalar windows are performed simultaneously within the PE array.

(2) In high-bandwidth setups – depicted in the middle and right sides of Figure 4 – both vertical and fully unrolled strategies are feasible. For vertical unrolling, points are stored in off-chip memory in an order that reflects the number of PEs. For example, with an HBM featuring 16 banks, each bank stores points in a staggered sequence such that bank0 contains $[P_0, P_{16}, P_{32}, \dots]$, bank1 contains $[P_1, P_{17}, P_{33}, \dots]$, etc. This staggered sequence of points allows us to keep the loading of scalars linearly $(s_0, s_1, s_2, ...)$ from PCIe without reordering. The main advantage of this arrangement is the concurrent processing of multiple scalars across PEs, leveraging the parallelism provided by the memory banks.

(3) In the case of fully unrolled processing (Figure 4 right), vertical unrolling is combined with horizontal unrolling, allowing computations to be performed on multiple points and scalars simultaneously. This maximizes parallelism by utilizing both inter-scalar and intra-scalar parallelization, achieving the highest throughput at the cost of increased hardware complexity.

Mapping MSM Workloads to Chiplets: Our chiplet-based architecture maps MSM workloads by distributing PEs and mixed window sizes across multiple chiplets to achieve scalability and efficiency. Figure 5 illustrates this mapping for vertical unrolling, showing the even distribution of PEs between two chiplets. Specifically, we employ a mixed window size configuration consisting of 6 PEs with 12-bit windows and 14 PEs with 13-bit windows, perfectly splitting the 253-bit scalar (adjusted from 254 bits using signed scalar reduction [22], [23]).

This configuration optimally balances PE area and latency, ensuring efficient utilization of hardware resources. Each chiplet houses 10 PEs, keeping the total chiplet area within approximately 80 mm^2 in the targeted TSMC 28nm library. This area constraint ensures scalability by maintaining manufacturing yields while avoiding excessive power and thermal constraints. By adopting this two-chiplet configuration, our architecture achieves high parallelism and throughput without compromising on area or power efficiency.

Optimizing Inter-Chiplet Communication: In 2.5D SiP, chiplets are connected through a silicon interposer using UCIe [24] interconnects. We optimize the chiplet-to-chiplet (C2C) communication requirement by tailoring the workload decomposition for the unique data flow of the MSM. In our design, we reduce the C2C communication requirement to the minimum: only one elliptic curve point per cycle. We achieve this by passing and processing the elliptic curve points serially within each chiplet. Each point passes through all the PEs within a particular chiplet before being transmitted to the adjacent chiplet, as shown in Figure 5. The serial pipeline offers computation-communication parallelism, allowing processing

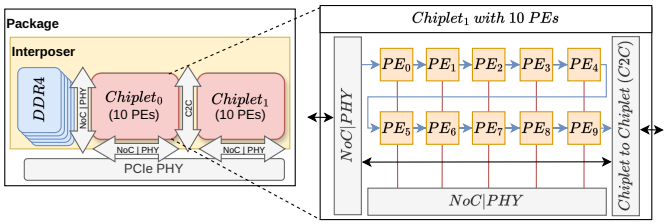

Fig. 5. Overview of an horizontal unrolled architecture with a two-chiplet configuration each incorporating 10 PEs.

elements to perform computations while points are transferred simultaneously.

This communication strategy adapts to different memory configurations to optimize throughput. For low-bandwidth DDR environments, horizontal unrolling reduces redundant data loading by ensuring each point is fetched only once, thereby minimizing off-chip memory interactions. In contrast, high-bandwidth HBM environments leverage vertical unrolling, which takes advantage of the parallel memory banks in HBM to enable simultaneous access to multiple points, maximizing memory bandwidth utilization. By aligning communication strategies with memory configurations, our design reduces inter-chiplet overhead and ensures scalability and efficient resource allocation across diverse environments.

IV. Evaluation and Results

This section presents performance results for our single- and multi-PE chiplet design based on TSMC 28nm technology. We additionally compare our area, power, latency, and scalability results with state-of-the-art MSM works to highlight the efficiency and flexibility of our design.

A. Single-PE Performance

Our single PE design demonstrates improvements in speed, area, and power compared to state-of-the-art MSM accelerators. The single PE with a window size of 13 bits occupies 7.11 mm^2 and consumes 4.6 W as shown in Table II.

Compared to GPU-based accelerators cuZK [2] and GZKP [25], our single-PE design achieves speed improvements between 1.9× and 5.3× depending on N. Compared to FPGA-based implementations such as CycloneMSM [6], Hardcaml [7], and BSTMSM [26], our design achieves a speedup of 7.1×, 6.5×, and 3.4× at $N = 2^{23}$, respectively. We emphasize our three to four times higher clock frequency than those works (1 GHz compared to 250–300 MHz, as reported in Table II). When normalizing our ASIC results to their FPGA results we still achieve slight speedups of up to 1.8×.

It is also noteworthy, that these FPGA designs utilize nearly all available logic resources, leaving little room for scalability, whereas our ASIC architecture is specifically optimized for efficient scaling with additional PEs. Gypsophila [12] presents FPGA results for single PE and ASIC results for multiple PEs. Our single PE design achieves a comparable performance to their FPGA-based results.

TABLE II
PERFORMANCE COMPARISON WITH RELATED WORKS

Work	PEs	Platform	Freq. MHz	Area / Resources mm^2 or LUT/FF/DSP/BRAM	Power W	Latency in ms for various N 2^16	2^17	2^18	2^19	2^20	2^21	2^22	2^23	2^24	2^25	2^26
GZKP [25]	1	Nvidia V100	-	-	-	7	-	20	-	62	-	240	-	1,100	-	4,000
cuZK [2]	1	Nvidia V100	-	-	-	-	-	-	27	47	90	171	312	-	-	-
PipeMSM [19]	1	Xilinx U55C	125	-	34.9	17.6	35.9	68.8	136.6	273.0	-	-	-	-	-	-
CycloneMSM [6]	1	AWS F1	250	526k / 661k / 2,277 / 623	43.5	-	-	-	-	-	-	817.9	1,199.0	1,761.0	3,016.0	5,656.0
Hardcaml [7]	1	Xilinx VU9P	278	387k / 733k / 2,999 / 883.5	-	-	-	-	499.0	540.0	620.0	780.0	1,094.0	-	-	-
BSTMSM [26]	1	Xilinx U250	300	410k / 744k / 2,920 / 623	-	-	-	23.0	40.0	75.0	145.0	285.0	564.0	1,124.0	2,242.0	4,479.0
Gypsophila [12]	1	Xilinx VU13P	200	1,459k / - / 3,684 / -	48.9	-	-	23.8	44.8	86.8	170.7	338.6	674.3	1,346.0	2,689.0	-
PipeZK [11]	1	UMC 28nm	300	16.86	2.4	22.0	45.0	92.0	184.0	-	-	-	-	-	-	-
PriorMSM [13]	1	TSMC 28nm	1,000	9.21	5.3	1.8	3.3	6.2	12.0	24.0	47.0	95.0	189.0	377.0	754.0	1,509.0
Gypsophila* [12]	16	TSMC 12nm	1,000	79.80	45.3	-	-	4.8	9.0	17.4	34.2	67.8	135.0	269.5	538.4	-
Ours	**1**	**TSMC 28nm**	**1,000**	**7.11**	**4.6**	**1.31**	**2.62**	**5.24**	**10.49**	**20.97**	**41.94**	**83.89**	**167.77**	**335.55**	**671.09**	**1,342.18**
Ours	**5**	**TSMC 28nm**	**1,000**	**35.54**	**23.0**	**0.26**	**0.52**	**1.05**	**2.10**	**4.19**	**8.39**	**16.78**	**33.55**	**67.11**	**134.22**	**268.44**
Ours	**10**	**TSMC 28nm**	**1,000**	**71.09**	**46.1**	**0.13**	**0.26**	**0.52**	**1.05**	**2.10**	**4.19**	**8.39**	**16.78**	**33.55**	**67.11**	**134.22**
Ours	**20**	**TSMC 28nm**	**1,000**	**2×71.09**	**92.2**	**0.07**	**0.13**	**0.26**	**0.52**	**1.05**	**2.10**	**4.19**	**8.39**	**16.78**	**33.55**	**67.11**

* Performs 16 independent MSMs in parallel. Hence, the latency for one MSM is identical to single PE case but throughput is approx. 16× higher than in single PE.

TABLE III
THROUGHPUT COMPARISON WITH GYPSOPHILA

Work	PEs	Platform	Freq. MHz	Area / Resources mm^2	Power W	Latency in ms for various N 2^16	2^17	2^18	2^19	2^20	2^21	2^22	2^23	2^24	2^25	2^26
Gypsophila* [12]	16	TSMC 12nm	1,000	79.80	45.3	-	-	3,328	1,777	919	468	236	119	59	30	-
Ours	**20**	**TSMC 28nm**	**1,000**	**2×71.09**	**92.2**	**15,240**	**7,625**	**3,813**	**1,907**	**954**	**447**	**238**	**119**	**60**	**30**	**15**

* Performs 16 independent MSMs in parallel. Hence, the latency for one MSM is identical to single PE case but throughput is approx. 16× higher than in single PE.

We now consider single-PE works for ASIC. Compared to PipeZK [11], we achieve a 17× speedup in our single-PE case with similar technology nodes. Moreover, our design is 2.4× smaller but needs 1.9× more power. Considering PriorMSM [13], we reach a moderate speedup of up to 1.37× while requiring 1.3× less area and 1.15× less power. This shows the competitiveness of our single PE design with related ASIC solutions for MSM.

B. Multi-PE and Multi-Chiplet Scaling

Increasing the number of processing elements (PEs) in our design significantly accelerates MSM operations by enabling parallel processing of multiple windows. Configurations with 5, 10, and 20 PEs and a mixture of PEs with 12- and 13-bit window sizes show consistent improvements in latency and throughput, as reported in Table II. The area and power usage increase linearly with the number of PEs meaning that each additional PE proportionally reduces the latency of MSM while increasing area utilization. For instance, the 10-PE configuration, occupying approximately 71.09 mm² and consuming 46.1 W of power (Table II), reduces the required computation passes to just 2 — 10× fewer than the 20 passes needed in a single-PE setup. Similarly, the 20-PE configuration, spanning two chiplets with 10 PEs each, achieves the theoretical minimum of 1 pass per MSM operation. Our chiplet-based approach with horizontal unrolling – where each chiplet has 71.09 mm^2 – allows higher production yield than a monolithic chip with up to 269 mm^2.

The only work published that uses multiple PEs is Gypsophila [12]. However, Gypsophila uses a different approach than our work and computes 16 independent MSMs in parallel. Thereby, each of the 16 PEs computes one MSM. Thus, Gypsophila shows an almost identical MSM latency in the 16 PE case compared to the single PE case. Since our 20 PEs jointly compute one MSM, we compare the overall throughput in MSMs performed per second. The according benchmark is presented in Table III. Compared to Gypsophila, we achieve a moderately higher throughput for small N and a similar throughput for larger N. To compare to the area consumption of Gypsophila, we scale our area from the 28nm to the 12nm technology by applying a factor of 0.25 [27], [28]. Based on that, our work has an area benefit of 2.2×. This benefit of our work is explained by our optimal window sizes of 12 to 13 bits whereas Gypsophilahas 16-bit windows. In addition, our chiplet-based approach allows lower production costs and higher yield.

V. CONCLUSION

We presented a chiplet-based MSM accelerator to address the high computation and data demands in ZKP while offering scalability, low production cost, and increased yield. By introducing a memory-aware PE design and leveraging mixed window size configurations, we optimized the trade-offs between area, power, and computational throughput. Our design employs flexible workload distribution strategies, including horizontal, vertical, and fully unrolled approaches, tailored to different memory configurations for maximum efficiency. We further minimized inter-chiplet communication overhead through MSM's data flow-specific customization to the workload distribution and interconnect strategies. Experimental evaluations demonstrated that our single-PE design achieved a 1.37x speedup and a 1.3x area reduction over prior works, while the multi-PE chiplet design outperformed monolithic counterparts by 2.2x in area-time product.

REFERENCES

[1] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, ser. STOC '85. New York, NY, USA: Association for Computing Machinery, 1985, p. 291–304. [Online]. Available: https://doi.org/10.1145/22145.22178

[2] T. Lu, C. Wei, R. Yu, C. Chen, W. Fang, L. Wang, Z. Wang, and W. Chen, "Cuzk: Accelerating zero-knowledge proof with a faster parallel multi-scalar multiplication algorithm on gpus," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2023, no. 3, pp. 194–220, 2023.

[3] J. Groth, "On the size of pairing-based non-interactive arguments," in *Advances in Cryptology – EUROCRYPT 2016*, M. Fischlin and J.-S. Coron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 305–326.

[4] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge," Cryptology ePrint Archive, Paper 2019/953, 2019. [Online]. Available: https://eprint.iacr.org/2019/953

[5] A. Chiesa, Y. Hu, M. Maller, P. Mishra, P. Vesely, and N. Ward, "Marlin: Preprocessing zkSNARKs with universal and updatable SRS," Cryptology ePrint Archive, Paper 2019/1047, 2019. [Online]. Available: https://eprint.iacr.org/2019/1047

[6] K. Aasaraai, D. Beaver, E. Cesena, R. Maganti, N. Stalder, and J. Varela, "FPGA acceleration of multi-scalar multiplication: CycloneMSM," Cryptology ePrint Archive, Paper 2022/1396, 2022. [Online]. Available: https://eprint.iacr.org/2022/1396

[7] A. Ray, B. Devlin, F. Y. Quah, and R. Yesantharao, "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*, ser. FPGA '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 33–39. [Online]. Available: https://doi.org/10.1145/3626202.3637577

[8] ZPrize, "Accelerating the future of zero knowledge cryptography," 2023. [Online]. Available: https://https://www.zprize.io/

[9] N. Pippenger, " On the evaluation of powers and related problems ," in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 1976, pp. 258–263. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SFCS.1976.21

[10] G. Luo, S. Fu, and G. Gong, "Speeding up msm over fixed points towards efficient zksnarks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2023, no. 2, p. 358–380, Mar. 2023. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/10287

[11] Y. Zhang, S. Wang, X. Zhang, J. Dong, X. Mao, F. Long, C. Wang, D. Zhou, M. Gao, and G. Sun, "Pipezk: Accelerating zero-knowledge proof with a pipelined architecture," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 416–428.

[12] C. Liu, H. Zhou, L. Yang, J. Xu, P. Dai, and F. Yang, "Gypsophila: A scalable and bandwidth-optimized multi-scalar multiplication architecture," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3658259

[13] C. Liu, H. Zhou, P. Dai, L. Shang, and F. Yang, "Priormsm: An efficient acceleration architecture for multi-scalar multiplication," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 29, no. 5, Aug. 2024. [Online]. Available: https://doi.org/10.1145/3678006

[14] X. Ma, Y. Wang, Y. Wang, X. Cai, and Y. Han, "Survey on chiplets: interface, interconnect and integration methodology," *CCF Transactions on High Performance Computing*, vol. 4, pp. 43 – 52, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:247846633

[15] S. Kim, J. Kim, J. Choi, and J. H. Ahn, "Cifher: A chiplet-based fhe accelerator with a resizable structure," 2024. [Online]. Available: https://arxiv.org/abs/2308.04890

[16] A. Aikata, A. C. Mert, S. Kwon, M. Deryabin, and S. S. Roy, "REED: Chiplet-based accelerator for fully homomorphic encryption," Cryptology ePrint Archive, Paper 2023/1190, 2023. [Online]. Available: https://eprint.iacr.org/2023/1190

[17] Y. Du, Y. Wang, B. Li, F. Li, S. Liang, H. Li, X. Li, and Y. Han, "Chiplever: Towards effortless extension of chiplet-based system for fhe," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3657321

[18] D. Bernstein, "Explicit-formulas database," 2024. [Online]. Available: https://cir.nii.ac.jp/crid/1570572699314327808

[19] C. F. Xavier, "PipeMSM: Hardware acceleration for multi-scalar multiplication," Cryptology ePrint Archive, Paper 2022/999, 2022. [Online]. Available: https://eprint.iacr.org/2022/999

[20] P. Barrett, "Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor," in *Advances in Cryptology — CRYPTO' 86*, A. M. Odlyzko, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 311–323.

[21] D. M. Gordon, "A survey of fast exponentiation methods," *Journal of Algorithms*, vol. 27, no. 1, pp. 129–146, 1998. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0196677497909135

[22] A. Rezai and P. Keshavarzi, "An efficient scalar multiplication algorithm for elliptic curve cryptography using a new signed-digit representation," in *Networking and Communication Conference*, 12 2013, pp. 44–48.

[23] P. Balasubramaniam and E. Karthikeyan, "Elliptic curve scalar multiplication algorithm using complementary recoding," *Applied Mathematics and Computation*, vol. 190, no. 1, pp. 51–56, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0096300307000276

[24] UCIe, "For the first time, ucie shares bandwidth speeds between chiplets," 2023. [Online]. Available: https://www.hpcwire.com/2023/06/07/for-the-first-time-ucie-shares-bandwidth-speeds-between-chiplets/

[25] W. Ma, Q. Xiong, X. Shi, X. Ma, H. Jin, H. Kuang, M. Gao, Y. Zhang, H. Shen, and W. Hu, "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*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 340–353. [Online]. Available: https://doi.org/10.1145/3575693.3575711

[26] B. Zhao, W. Huang, T. Li, and Y. Huang, "Bstmsm: A high-performance fpga-based multi-scalar multiplication hardware accelerator," in *2023 International Conference on Field Programmable Technology (ICFPT)*, 2023, pp. 35–43.

[27] T. VLSI, "Tsmc 7nm, 16nm and 28nm technology node comparisons," 2021. [Online]. Available: https://teamvlsi.com/2021/09/tsmc-7nm-16nm-and-28nm-technology-node-comparisons.html

[28] L. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, Jul. 2016, publisher Copyright: © 2016 The Authors.