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

Florian Hirner, Florian Krieger, and Sujoy Sinha Roy *Senior Member, IEEE*

***Abstract*—This paper presents a high-performance architecture for accelerating Multi-Scalar Multiplication (MSM) on ASIC platforms, targeting cryptographic applications with high throughput and scalability demands. Current MSM accelerators on FPGA and ASIC platforms typically focus on designing efficient processing elements (PEs) to perform resource-intensive elliptic curve point operations, which require a high number of 384-bit modular multipliers. Our approach diverges from existing works by adopting a chiplet-based design, which optimally balances area, power consumption, and computational throughput. By analyzing memory requirements across window sizes, we determine an optimal mixed configuration of 12- and 13-bit windows, which allows efficient integration of multiple PEs per chiplet. Considering the single-PE case, our design achieves a 1.37x speedup and a 1.3x area reduction over prior works. Moreover, our multi-PE chiplet design outperforms monolithic designs by 2.2x in area-time product while allowing lower production costs and higher yield.**

***Index Terms*—Multi-Scalar Multiplication, Zero-Knowledge Proof, 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], [2]. As shown in Figure 1, MSM accounts for over 70% of proof generation time [3]. Its computational dominance increases with the number of points N, as seen in systems like Groth16 [4], PlonK-KZG [5], and Marlin-KZG [6], which process tens to hundreds of gigabytes of data and take minutes for 2^{25} to 2^{28} point MSM. Therefore, MSM requires careful optimization of resource utilization, memory bandwidth, and overall latency. Only highly optimized designs enable practical and performant ZKP applications in areas such as blockchain systems and privacy-preserving cryptographic protocols [7], [8].

CycloneMSM [9] and Hardcaml [10] are high-performance FPGA-based MSM accelerators from the 2023 Z-Prize [11] challenge. These designs leverage Pippenger's bucket method [12] to accelerate MSM by decomposing the computation into three key steps: bucket classification, bucket aggregation, and result aggregation [7]. 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 (PE) even in high-end FPGAs. OPTIMSM [13] builds on prior mentioned work and augments Pippenger with efficiently computable endomorphisms, precomputed point multiples, signed-digit reduction [14], [15], and a novel iteration technique. These techniques allow for them to compute larger windows without exhausting on-chip memory of the FPGA.

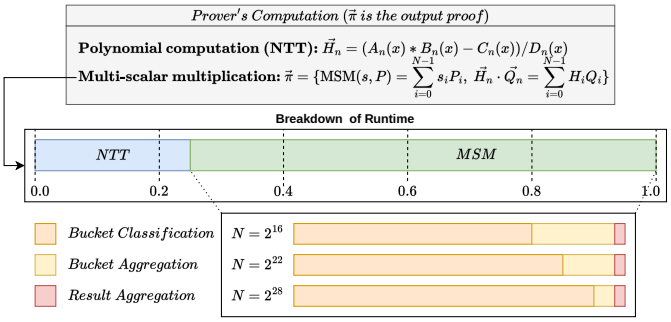


Fig. 1. Overview of workload distribution for NTT, MSM, and Bucketing operations involved in proof generation of paring-based ZKPs.

Compared to FPGA, ASIC architectures such as PipeZK [16], Gypsophila [17], and PriorMSM [18] primarily employ single processing element (PE) configurations with optimized memory access strategies to mitigate bandwidth pressure from bucket collisions during classification. Among these, only Gypsophila [17] 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 III, 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 [19]. Recent works, such as CiFHer [20], REED [21] and Chiplever [22], have explored the adoption of chiplet-based architectures for Fully Homomorphic Encryption (FHE), which like MSM is also computation

F. Hirner, F. Krieger, S. S. Roy are with Institute of Information Security, Graz University of Technology, Graz, Austria (e-mail: {florian.hirner, florian.krieger, sujoy.sinharoy}@tugraz.at).

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

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. We the analysis of window sizes to propose different chiplet-based architectures that use either vertical, horizontal, partial or full unrolling.
- **Inter-chiplet communication:** We propose multiple workload distribution and interconnect strategies to minimize inter-chiplet and off-chip memory communications. We use these distribution strategies (vertical, horizontal, partial or full unrolling) in our chiplet-based architectures.
- **Comprehensive evaluation:** We evaluate our design across varying configurations, showing up to $1.14\times$ and $2.2\times$ 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.

This paper is organized as follows: Section II reviews the background on elliptic curve arithmetic, MSM, the Pippenger algorithm, and monolithic versus chiplet-based ASIC designs. Section III analyzes processing element architectures and window-size tradeoffs, identifying a memory-aware mixed-window configuration. Section IV presents scalable chiplet-based MSM architectures with communication-efficient workload distribution. Section V evaluates the proposed designs, demonstrating improved throughput and area efficiency compared to state-of-the-art monolithic ASIC implementations.

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. We explain the essentials to understand the core computational principles and architectural choices for MSM. Additionally, we contrast monolithic and chiplet-based designs, discussing the trade-offs in terms of scalability, yield, and communication, which guide our approach to achieving high performance and modularity in MSM acceleration.

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 s-times accumulation of P.

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 [23].

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 around 256-bit $s_i \in \mathbb{F}_p \subset \mathbb{N}$ and points $P = (P_0, P_1, \ldots, P_{N-1})$ on an elliptic curve E, MSM is expressed as $\text{MSM}(s, P) = s_0 \cdot P_0 + s_1 \cdot P_1 + \cdots + s_{N-1} \cdot P_{N-1}$. Given the computational intensity of MSM, efficient algorithms like the *Pippenger algorithm* [12] are crucial for minimizing the number of scalar multiplications and achieving high-performance hardware implementations. In zero-knowledge proofs N can easily reaches 2^{26} for larger circuits, therefore, the computational cost of MSM is substantial which driving the need for efficient algorithms and hardware acceleration.

C. Pippenger Algorithm

The Pippenger algorithm [12], 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 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,

Algorithm 1 Pippenger Algorithm - Bucket Method [17]

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

it computes $S_j = \sum_{k=0}^{2^w-1} k \cdot B_{j,k}$. disIn 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.

D. Point Addition in Pippenger-Based MSM

As described in Section II-C, the Pippenger algorithm decomposes MSM into bucket classification, bucket aggregation, and result aggregation. Bucket classification mainly involves point additions and can be efficiently pipelined with appropriate collision handling [9], whereas bucket aggregation and result aggregation require both point additions and point doublings and are dominated by data dependencies. In particular, the accumulation steps $S_j \leftarrow S_j + k \cdot B_{j,k}$ and $R \leftarrow R + S_j$ rely on repeated, sequential point operations that must wait for previously computed values. As a result, elliptic-curve point addition is a key operation in Pippenger-based MSM, as the algorithm requires both point additions and point doublings depending on the relation between input operands. The computational cost of these operations depends on the underlying curve model and coordinate representation, which is discussed in the next section.

E. Different Point Adder Architectures

Prior hardware implementations [9], [10], [13], [18], [24] of Pippenger-based MSM adopt different point adder architectures depending on how point addition and point doubling are

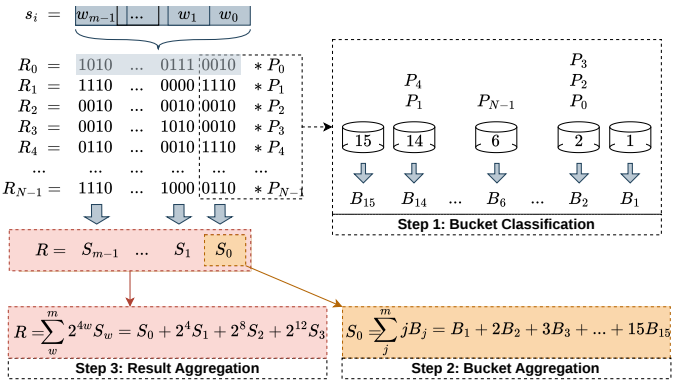


Fig. 2. Illustrative overview of the classification and accumulation steps in the Pippenger algorithm for computing R.

supported within the algorithmic flow. A common distinction is between *mixed-coordinate* adders and *full* point adders.

Mixed-coordinate adders exploit the use of different coordinate representations for the two input operands, for example representing incoming points in standard projective coordinates while storing bucket elements in extended coordinates. This situation naturally arises during bucket classification and early bucket aggregation, where the operands are guaranteed to be distinct. As a result, point doubling cannot occur, allowing mixed adders to omit doubling-specific arithmetic and reduce the number of finite-field operations. A representative mixed-coordinate addition schedule in extended Twisted Edwards coordinates is summarized in Table I. Table I shows a commonly used computation schedule of mixed point addition that uses four parallel computation units and takes a total of four steps.

In later stages of the Pippenger algorithm, particularly during bucket aggregation across k and result aggregation across windows j, all points reside in the same coordinate system and additions between identical points may occur. These operations implicitly require support for point doubling, i.e., the case when the addends happen to be equal. Since mixed-coordinate adders do not natively support point doubling, we have to first transform both addends to the same coordinate system, and then perform a full addition. Because of this reason, prior works [9] "emulate" full point operations through multiple passes by reusing modular arithmetic components of a mixed-adder datapath. Full point adders, in contrast, implement unified formulas that support both point addition and point doubling within a single coordinate system. Although this increases the arithmetic cost per operation, it enables each accumulation step to be executed in a single pass. An example of such a unified addition in extended Twisted Edwards coordinates is shown in Table II.

Overall, this distinction reflects a common tradeoff in Pippenger-based MSM accelerators: mixed-coordinate adders reduce the cost of individual operations but require additional scheduling to handle aggregation phases, whereas full adders increase per-operation cost while simplifying control flow during bucket and result aggregation.

F. Monolithic vs. Chiplet Designs

A fundamental architectural decision in modern large-scale ASIC design concerns the choice between monolithic and

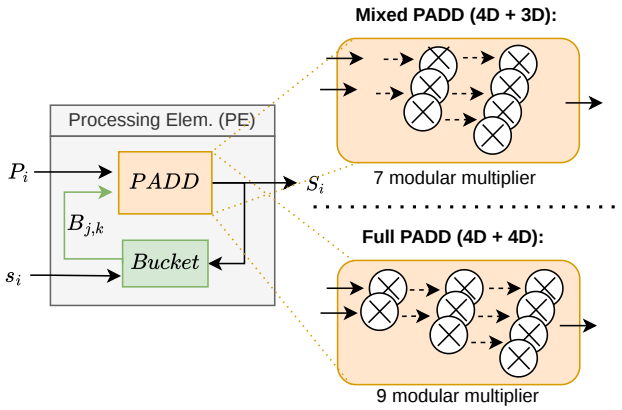


Fig. 3. Overview of a Processing Element (PE) with mixed (top) and full (bottom) PADD modules.

TABLE I
MixedAdd IN EXTENDED TWISTED EDWARDS COORDINATES [25]

Arithmetic Cost	Step	Computations on Parallel Processing Units			
2 add + 2 sub	1	$R_1 \leftarrow Y_1 - X_1$	$R_2 \leftarrow Y_2 - X_2$	$R_1 \leftarrow Y_1 + X_1$	$R_2 \leftarrow Y_2 + X_2$
3 mul + 1 add	2	$A \leftarrow R_1 \cdot R_2$	$B \leftarrow R_2 \cdot R_4$	$C \leftarrow T_1 \cdot u_2$	$D \leftarrow Z_1 + 1_2$
2 add + 2 sub	3	$E \leftarrow B - A$	$F = D - C$	$G \leftarrow D + C$	$H \leftarrow B + A$
4 mul	4	$X_3 \leftarrow E \cdot F$	$Y_3 \leftarrow G \cdot H$	$T_3 \leftarrow E \cdot H$	$Z_3 \leftarrow F \cdot G$

chiplet-based integration, each with distinct implications for performance, scalability, and manufacturing efficiency. Monolithic designs integrate all compute logic, memory structures, and interconnects into a single die. This approach has traditionally been favored for its simplicity and low-latency on-chip communication. Consequently, all prior ASIC-based MSM accelerator works, like [17], adopt monolithic designs and focus on architectural exploration on one die.

However, as acceleration demands grow with increasingly large MSM problem sizes, monolithic integration faces scalability limitations. Increasing die area diminishes the manufacturing yield and increases the fabrication costs superlinearly with area. This makes monolithic architectures less ideal for scaling with increasing application constraints in MSM computation. Chiplet-based System-in-Package (SiP) architectures have emerged as a general solution to these limitations in large and complex ASIC systems. Instead of fabricating a single large die, the large architecture design is partitioned into multiple smaller dies that are manufactured independently. Later the dies are integrated within a single package. This modular approach improves yield and production costs, and enables more flexible scaling. Chiplet-based designs are already widely deployed in large-scale CPUs, GPUs, and AI accelerators, and are increasingly being explored for computation- and memory-intensive cryptographic workloads, such as fully homomorphic encryption (FHE), as demonstrated by CiFHER [20], REED [21], and Chiplever [22].

A key challenge in chiplet-based systems is inter-chiplet communication. Data transfers acress chiplet boundaries incur higher latency and energy than on-die transfers in a monolithic chip, making efficient interconnect design and workload partitioning critical for chiplet decomposition. Modern chiplet platforms therefore rely on high-bandwidth interposers and structured interconnects, such as mesh or ring topologies,

TABLE II
FullAdd IN EXTENDED TWISTED EDWARDS COORDINATES [25]

Arithmetic Cost	Step	Computations on Parallel Processing Units			
2 add + 2 sub	1	$R_1 \leftarrow Y_1 - X_1$	$R_2 \leftarrow Y_2 - X_2$	$R_3 \leftarrow Y_1 + X_1$	$R_4 \leftarrow Y_2 + X_2$
3 mul + 1 add	2	$A \leftarrow R_1 \cdot R_2$	$B \leftarrow R_2 \cdot R_4$	$R_5 \leftarrow T_1 \cdot T_2$	$R_6 \leftarrow Z_2 + Z_2$
2 mul	3	*idle*	*idle*	$C \leftarrow R_5 \cdot k$	$D \leftarrow R_6 \cdot Z_2$
2 add + 2 sub	4	$E \leftarrow B - A$	$F \leftarrow D - C$	$G \leftarrow D + C$	$H \leftarrow B + A$
4 mul	5	$X_3 \leftarrow E \cdot F$	$Y_3 \leftarrow G \cdot H$	$T_3 \leftarrow E \cdot H$	$Z_3 \leftarrow F \cdot G$

combined with careful workload partitioning to limit chiplet-to-chiplet communication overheads. We point the reader to [26] for further information on chiplet design methodologies, partitioning strategies, interconnect standards, and design automation flows for multi-die integration.

III. Impact Analysis of Different Window Size on Area, Resources and Latency

This section analyzes key design considerations for point addition in an MSM accelerator, focusing on how window size selection influences bucket memory requirements, aggregation latency, and processing element (PE) organization. While the point adder is a fundamental building block of the PE, the emphasis here is on architectural tradeoffs rather than detailed circuit design. The analysis covers both single-PE and fully unrolled multi-PE configurations which motivates the window sizes and aggregation strategies adopted in later sections.

A. Theoretical Analysis of Window Size Tradeoffs

To support our empirical window size selection, we developed a theoretical cost model analyzing the tradeoffs between latency, memory size, and area. The scalar s and window w size determines the number of passes $m = \lceil s/w \rceil$, and the number of buckets 2^w directly impacts the memory footprint.

$$\text{ATP}(w) = A_{\text{PE}}(w) \times m = (A_{\text{PADD}} + A_{\text{bucket}}(2^w)) \times \left\lceil \frac{s}{w} \right\rceil \quad (1)$$

We define the Area-Time Product (ATP) as a design objective function in Equation 1. Here, A_{PADD} is fixed, while A_{bucket} grows approximately linearly with 2^w. We observed a significant growth in total memory beyond $w = 13$. Figure 4 visualizes this tradeoff, showing that ATP reaches a minimum in the 11–13 bit range. This balance results from diminishing latency gains and increasing memory footprint for higher w. Choosing window sizes within a narrow range (e.g., 12- and 13-bit) ensures balanced aggregation latency across PEs and avoids bottlenecks during result aggregation. If window size variance are too larger (e.g., combining 10-bit and 16-bit windows) the resulting pipeline stalls that occur during classification would imbalanced processing times.

While our analysis is primarily technology- and window-centric, it has implications on architectural partitioning. For instance, minimizing the number of passes directly impacts how many PEs are needed when unrolling across chiplets. Selecting window sizes that align with powers-of-two division of the scalar bit width (e.g., 12-bit and 13-bit for 253 bits) helps simplify partitioning across processing elements and improves load balancing.

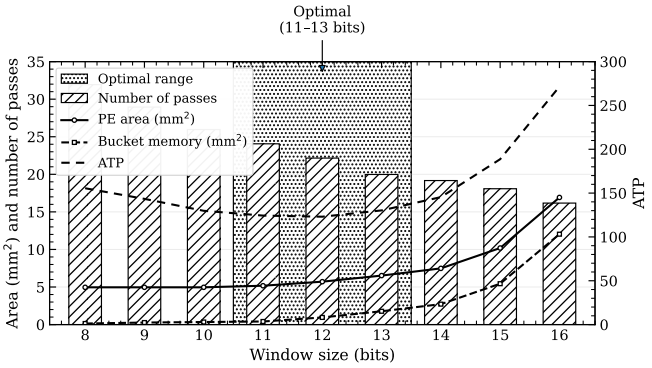


Fig. 4. Impact on area consumption, number of passes, and area-time product.

TABLE III
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 mm^2)	154	141	129	124	121	125	149	195	269
Power (in W)	122	111	99	91	84	76	72	68	61

B. 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 4 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 4), 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 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.

C. 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 $s_i = \langle w_0 | w_1 | \dots | w_{m-1} \rangle$. 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 III illustrates the trade-offs between area and power consumption for various window sizes ranging from 8 to 16 bits. Note that the power and area results reflect buckets using SRAM-based memories due to their large sizes. Therefore, the power scales logarithmically with the SRAM size. The first row of Table III 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 comparable smaller bucket memory as described in Sec III-B. 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 when considering the ATP. The optimal windows size range of multple PE is 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.

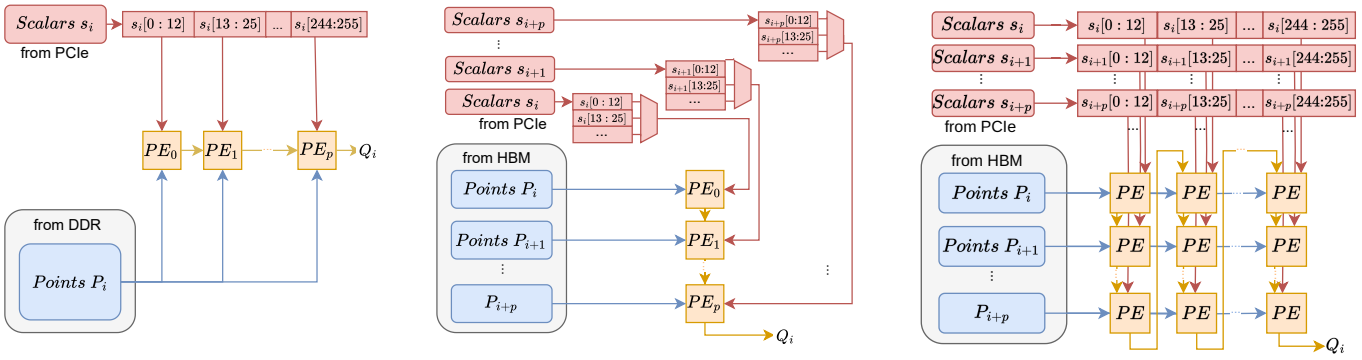
In terms of power consumption, Table III shows a decline for larger window sizes. This trend arises because, as window sizes grow, the memory footprint increases while the number of PEs decreases. Since the modular multipliers in PEs consume more power than memory, larger windows lead to reduced overall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

For example, a fully unrolled 16-bit window configuration, as used in [9], [17], requires a total area of 269 mm^2 and 61 W of power when using 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.

D. Impact of Window Size on Bucket Aggregation and Final Sum on Fully Unrolled Architecture with multi-PE case

After completing bucket classification, the points in each bucket must be aggregated to compute the final result of the bucket (Section II-C). 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 [27]. 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 fully unrolled architecture computes both classification and aggregation steps across PEs. Therein, all PEs process the same point P_i with different windows



(a) Horizontal Unrolled Architecture (b) Vertical Unrolled Architecture (c) Fully Unrolled Architecture

Fig. 5. Dataflow architecture illustrating scalar and point distribution across Processing Elements (PEs) for vertical, horizontal and full unrolling distribution techniques. Scalar values s_i are streamed from PCIe in contiguous batches, while point data P_i are retrieved from DDR or HBM memory.

$w_0 \dots w_{m-1}$. All PEs perform classification in parallel on different windows and finish at roughly the same time. After a PE finishes its classification, the PE independently aggregates its own buckets. During bucket aggregation, no inter-PE communication is required and all PEs run independently in parallel. In the fully unrolled case, the total latency therefore reduces from $m \times (L_{\text{class}} + L_{\text{agg}})$ in the sequential single-PE case to $(L_{\text{class}} + L_{\text{agg}})$. Whereas L_{class} denotes the classification latency for N points in one window and L_{agg} denotes the bucket aggregation latency for one window.

IV. Multi-PE Architecture

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 III. 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.

A. Distribution MSM Workloads Across PEs

We propose three 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 5 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** (as shown on the left side of Figure 5) 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** (as depicted as (b) and (c) in Figure 5) both vertical and fully unrolled strategies are feasible. For vertical unrolling, points in off-chip memory are distributed across banks, reflecting the number of PEs. For example, with an HBM featuring 16 banks, each bank stores points in a staggered sequence such that bank_0 contains $[P_0, P_{16}, P_{32}, \dots]$, bank_1 contains $[P_1, P_{17}, P_{33}, \dots]$. The main advantage of this arrangement is the concurrent processing of multiple scalars across PEs, leveraging the parallelism provided by access to multiple memory banks in HBM.

(3) **In the case of fully unrolled processing** (Figure 5 right), vertical unrolling is combined with horizontal unrolling, allowing computations to be performed on multiple points and scalars simultaneously. This maximizes parallelism by utilizing parallelization to achieving the highest throughput at the cost of increased hardware complexity.

B. 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. 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. Each chiplet houses 10 PEs, keeping the total chiplet area within approximately 80 mm^2 in the targeted TSMC 28nm library. By combining this chiplet approach with the MSM workload distribution from Section IV-A, we present two chiplet configurations tailored to different memory environments: a DDR-based horizontal unrolling and an HBM-based partial unrolling with dedicated HBM per chiplet.

1) DDR Configuration - Horizontal Unrolling: In the DDR configuration (Figure 6), all 20 PEs are arranged in a serial pipeline across two chiplets, where each PE is assigned a different window of the scalar. The points and scalars are loaded from DDR and PCIe, respectively. These are then streamed through the computation pipeline one at a time. The data access to off-chip memory is done in a linear streaming fashion to enable high throughput near the theoretical maximum of the DDR interface.

A simplified dataflow of the computation is illustrated in Figure 7. It illustrates a toy example with 2 chiplets, each

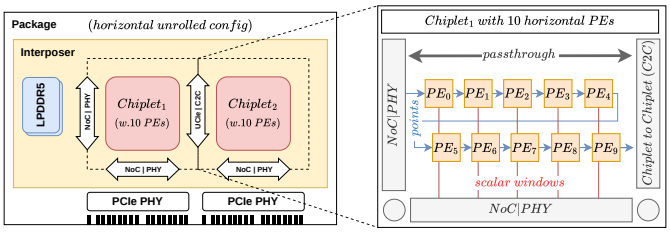


Fig. 6. Overview of the DDR-based horizontal unrolled architecture with a two-chiplet configuration, each incorporating 10 PEs in a serial pipeline.

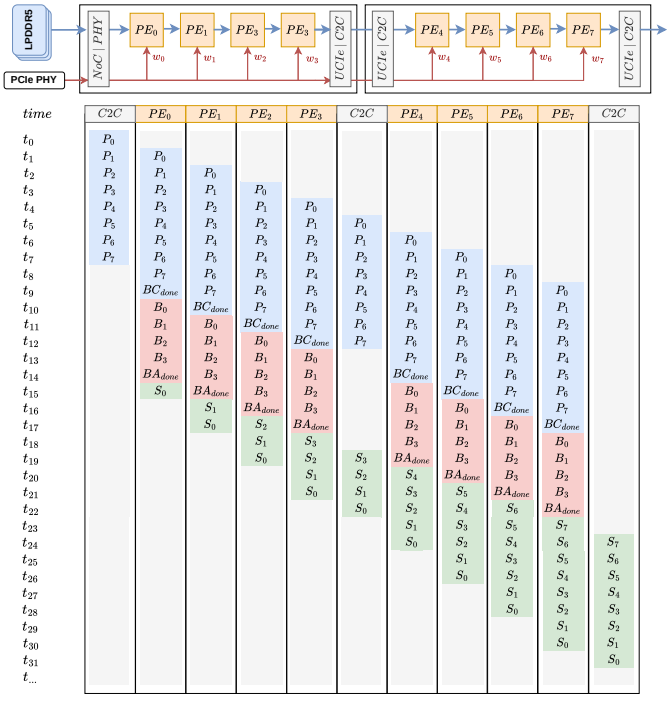


Fig. 7. Dataflow and workload distribution for the DDR-based horizontal unrolled architecture. Points flow through the PE pipeline one per cycle, with bucket classification (P_i), aggregation (B_k), and result forwarding (S_j).

consisting of 4 PEs per chiplet, and an MSM computation with 8 points. The bucket classification step is shown in blue, bucket aggregation in red, and result aggregation in green. At the beginning, each point P_i is loaded from DDR into Chiplet$_0$. All points P_i are then streamed from PE$_0$ to PE$_3$. Each PE performs bucket classification on its assigned window (w_0 through w_3) in case of Chiplet$_0$. All points are forwarded over the C2C link from Chiplet$_0$ to Chiplet$_1$. The C2C boundary incorporates a small buffer, which introduces a minor delay as shown in the center of Figure 7 as $C2C$. After passing the C2C boundary to Chiplet$_1$, the points continue through PE$_4$ to PE$_7$ to classify windows w_4 through w_7.

As shown in the timing diagram, all PEs operate in parallel with only a small pipeline fill delay. After all N points have been classified, each PE independently performs bucket aggregation on its local buckets depending on its assigned window. The partial bucket aggregation results S_j are then forwarded through the C2C link for the final result aggregation. The C2C link in this configuration requires enough bandwidth to transfer one scalar and one point per cycle. Therefore,

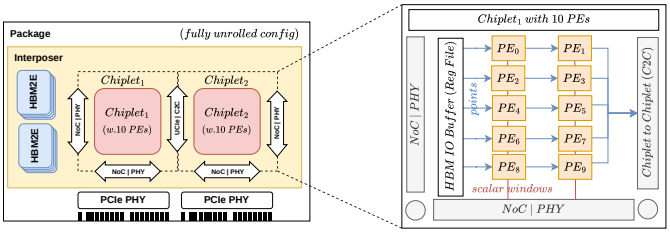


Fig. 8. Overview of HBM-based partially unrolled architecture with a two-chiplet configuration, each incorporating a 5×2 PE grid with dedicated HBM.

the bucket classification utilizes the full link capacity during the computation. In addition, the same C2C link is reused at the end of the bucket aggregation where the resulting S are collected before results aggregation. This results in a minimal inter-chiplet overhead. Note that for real workloads the classification step takes the majority of the computation time since it scales with N while the a aggregation scales with w. Therefore the aggregation takes a proportionally smaller share in the actual computation compared to our toy example.

2) HBM Configuration: Partial Unrolling: Therefore each chiplet processes two windows per horizontal pair. Note that our architecture uses two chiplets, therefore, we process four windows total in each computation pass. Therefore only 5 passes are required to cover all 20 windows. The two HBM2E stacks provide a combined 4096 bits per cycle, sufficient to load the required 5 points per cycle ($5 \times 768 = 3840$ bits). A small register file collects the HBM data and groups them into complete elliptic curve points. All 5 points first pass through the PEs of Chiplet$_0$ and are then forwarded over the C2C link to Chiplet$_1$. This is similar to the DDR configuration with 5 parallel point streams instead of only one. Note that each point is loaded from HBM only once per pass which computes four windows at once. This configuration results in an HBM bandwidth utilization of 78.1% (480 GB/s of 614 GB/s) and a C2C bandwidth utilization of 78.7% (496 GB/s of 614 GB/s), providing sufficient headroom for protocol overhead and scheduling inefficiencies.

Figure 9 illustrates the dataflow using a toy example with 2 chiplets, each consisting of 4 PEs in a 2×2 grid, and a MSM computation with 8 points. All points are loaded from the HBM connected to Chiplet$_0$. Each point passes through the PEs of Chiplet$_0$ first and is then forwarded over the C2C link to Chiplet$_1$. Within Chiplet$_0$, PE$_0$ and PE$_1$ form a horizontal pair processing P_0 through P_1 on different windows, while PE$_2$ and PE$_3$ form a second horizontal pair processing P_2 through P_3 on the same windows. The two horizontal pairs run in parallel, fed by separate HBM channels. Chiplet$_1$ operates identically on the same points but on different windows (w_4 through w_7). After classification and bucket aggregation, the PEs that share the same window across both chiplets produce partial results that must be combined (e.g., $S_0 = S_{0,0} + S_{0,1}$). Only these partial results are exchanged over the C2C link for the final result aggregation $R = \sum_j S_j \cdot 2^{j \cdot w}$.

C. Optimizing Inter-Chiplet Communication

In 2.5D SiP [28], chiplets are connected through a silicon interposer using UCIe [29] interconnects. We optimize the

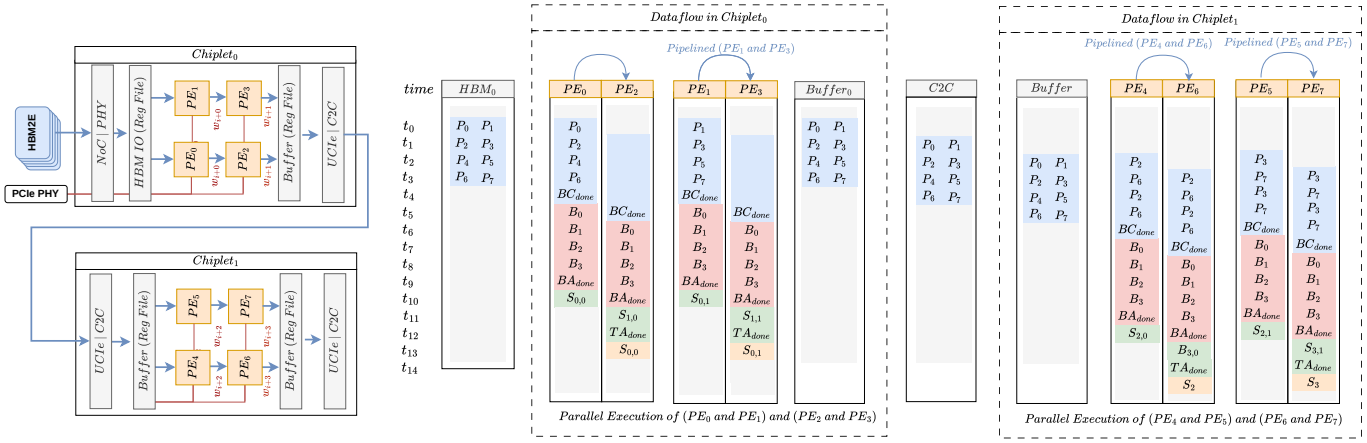


Fig. 9. Dataflow and workload distribution for the HBM-based partially unrolled architecture. Each chiplet loads five points per cycle from its dedicated HBM. Horizontal PE pairs process the same points on different windows, while vertical groups process different points on the same windows in parallel.

TABLE IV
C2C COMMUNICATION COST PER CONFIGURATION

Configuration	Bits/cycle	Energy/cycle	Power
DDR setting	896	448 pJ	448 mW
HBM setting	3968	1984 pJ	1984 mW

chiplet-to-chiplet (C2C) communication requirement by tailoring the workload decomposition for the unique data flow of the MSM. Therefore, we reduced the C2C communication to a minimum in our pipelined architecture, which requires a full point per cycle for processing. Note, that this only applies to the DDR configuration, while the HBM configuration requires five elliptic curve points per cycle. We achieve this reduced C2C communication by serially processing and streaming of the elliptic curve points. Each point passes through all the PEs within a particular chiplet before being transmitted to the adjacent chiplet, as shown in Figure 6.

D. Analysis of Power Consumption for C2C Communication

We estimate the peak target C2C power consumption PP using Eq. 2, where b is the number of bits transferred per cycle, f is the operating frequency, and E_{bit} is the interconnect energy per bit. We use $E_{\text{bit}} = 0.5$ pJ/bit for standard packaging as reported in [30].

$$PP_{\text{C2C}} = b \times f \times E_{\text{bit}} \tag{2}$$

In the DDR configuration, the C2C link transfers one point and the scalar windows to Chiplet$_1$ per cycle, totaling $b = 768 + 128 = 896$ bits. In the HBM configuration, five points and the corresponding scalar window slices per cycle, totaling $b = 5 \times 768 + 128 = 3968$ bits. The resulting energy and power estimates are summarized in Table IV.

E. Scalability Analysis

The proposed chiplet layout is modular and can be scaled by adding additional chiplets. As explained in Section II-C, the number of passes is given by $\lceil m/(2 \cdot C) \rceil$, where m is the total number of windows and C is the number of chiplets. For our design with $m = 20$, a 2-chiplet configuration requires 5 passes while a 10-chiplet configuration computes all 20 windows in a single pass. When scaling to more chiplets, each additional chiplet in the forwarding chain introduces a small data delay as points have to travel from chiplet to chiplet and propagate through the pipeline. This small warmup delay to fill the pipeline is in the range of 100 cycles per chiplet, and becomes negligible relative to the classification of $N \geq 2^{16}$ points. Therefore, assuming a proportional latency reduction for an increased chiplet count is reasonable.

Our scaling approach is analogous to how AMD/Xilinx Alveo FPGAs scales multiple chiplets [31] called Super Logic Regions (SLRs). For example, the C1100 FPGA [32] uses two SLRs and the U280 FPGA [33] uses three SLRs. In all cases, the HBM is connected to the first SLR (chiplet) and data flows from SLR to SLR via the SLR-crossing interconnect, which we took as inspiration for our chiplet-to-chiplet forwarding strategy. We simulated this data forwarding approach on an Alveo U280, where the SLR-crossing serves as a abstraction for the UCIe-based C2C link in our ASIC design.

V. 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. Evaluation Methodology

The area and power results reported in this work are obtained through logic synthesis using Cadence Genus 2019.11 targeting a TSMC 28nm standard cell library. The SRAM area for the bucket memories is estimated using an SRAM compiler for the same technology node while HBM2E bandwidth and inter-chiplet communication estimates are based on published specifications [30], [36]. Our evaluation is based on the same methodology as comparable works such as Gypsophila [17], REED [21], and CiFHEr [20] that analyze their results at the synthesis and architectural modeling level. In addition we follow a similar comparison model as in the literature [3],

TABLE V
PERFORMANCE COMPARISON WITH RELATED WORKS

Work	PEs	Chiplets	Platform	Freq.	Area / Resources	Power	Latency in ms for various N										
				MHz	mm^2 or LUT/FF/DSP/BRAM	W	2^16	2^17	2^18	2^19	2^20	2^21	2^22	2^23	2^24	2^25	2^26
GPU-Based Works																	
GZKP [34]	1	-	Nvidia V100	-	-	-	7	-	20	-	62	-	240	-	1,100	-	4,000
cuZK [3]	1	-	Nvidia V100	-	-	-	-	-	-	27	47	90	171	312	-	-	-
FPGA-Based Works																	
PipeMSM [24]	1	1	Xilinx U55C	125	-	34.9	17.6	35.9	68.8	136.6	273.0	-	-	-	-	-	-
CycloneMSM [9]	1	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 [10]	1	1	Xilinx VU9P	278	387k / 733k / 2,999 / 883.5	-	-	-	-	499.0	540.0	620.0	780.0	1,094.0	-	-	-
BSTMSM [35]	1	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 [17]	1	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	-
OPTIMSM [13]	1	1	Xilinx U55C	260	721K / 885K / 2,147 / 2,032	-	-	-	18.0	32.0	60.0	117.0	231.0	459.0	914.0	-	-
OPTIMSM [13]	4	4	4xXilinx U55C	260	721K / 885K / 2,147 / 2,032	-	-	-	7.3	11.0	18.0	33.0	61.0	118.0	231,0	-	-
ASIC-Based Works																	
PipeZK [16]	1	1	UMC 28nm	300	16.86	2.4	22.0	45.0	92.0	184.0	-	-	-	-	-	-	-
PriorMSM [18]	1	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* [17]	16	1	TSMC 12nm	1,000	79.80	45.3	-	-	4.8	9.0	17.4	34.2	67.8	135.0	269.5	538.4	-
Our ASIC-Based Work in Different Configurations																	
Ours	**1**	**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**	**1**	**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**	**1**	**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**	**2**	**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.

[16]–[18], [35]. All comparisons use the same workload sizes $N = 2^{16}$ to 2^{26}. For GPU comparisons we focus on latency, while for FPGA and ASIC we additionally compare area and power. We discuss each platform separately below.

B. 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 V.

Compared to GPU-based accelerators cuZK [3] and GZKP [34], our single-PE design achieves speed improvements between 1.9× and 5.3× depending on N. Compared to FPGA-based implementations such as CycloneMSM [9], Hardcaml [10], and BSTMSM [35], 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 V). When normalizing our ASIC results to their FPGA results we still achieve slight speedups of up to 1.8×. OPTIMSM [13] augments the Pippenger algorithm by introducing endomorphism and precomputed point multiples. It reports 914ms for a 2^{24} MSM on one U55C and 231ms when scaling to four U55Cs. Note that they leverage the FPGA's on-chip BRAM/URAM capacity to store the large bucket array required in their design. In contrast, we avoid large windows (e.g., $w > 13$) since bucket memory grows exponentially with w, making this approach poorly scalable for SRAM-dominated ASIC designs.

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 [17] 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.

We now consider single-PE works for ASIC. Compared to PipeZK [16], 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 [18], 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.

C. 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 V. 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 mm^2 and consuming 46.1 W of power (Table V), 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 ASIC work that uses multiple PEs is Gypsophila [17]. However, Gypsophila uses a different approach

TABLE VI
THROUGHPUT COMPARISON WITH GYPSOPHILA [17]

Work	PEs	Chiplets	Platform	Freq.	Area / Resources	Power	Latency in ms for various N									
				MHz	mm^2	W	2^16	2^17	2^18	2^19	2^20	2^21	2^22	2^23	2^24	2^25	2^26
Gypsophila* [17]	16	1	TSMC 12nm	1,000	79.80	45.3	-	-	3,328	1,777	919	468	236	119	59	30	-
Ours	**20**	**2**	**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 MSM latency is identical to single PE case but throughput is approx. 16× higher.

TABLE VII
LOGIC VS. SRAM AREA BREAKDOWN AT 28NM

Configuration	SRAM area	Logic area	Total area	SRAM %
1 PE (13b)	0.754 mm^2	6.356 mm^2	7.11 mm^2	10.6%
10 PE (3×12b + 7×13b)	6.409 mm^2	64.681 mm^2	71.09 mm^2	9.0%
20 PE (6×12b + 14×13b)	12.818 mm^2	129.362 mm^2	142.18 mm^2	9.0%

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 VI. 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 [17], we scale our area from the 28nm to the 12nm technology by applying a factor of 0.25 [37], [38]. Based on that, our work has an area advantage of 2.2×. This benefit of our work is explained by our optimal window sizes of 12 to 13 bits whereas Gypsophila [17] has 16-bit windows. In addition, our chiplet-based approach allows lower production costs and higher yield.

Similar to configuration-aware FHE designs like REED [21] and CiFHEr [20], our work shows that while HBM bandwidth is essential to support high-throughput MSM, the achievable inter-chiplet bandwidth may become a limiting factor. Thus, interconnect-aware partitioning and load balancing are critical in chiplet-based ASIC deployments.

D. Technology Scaling: Logic vs. SRAM

The uniform 0.25× scaling factor used above applies well to logic, but SRAM may scale less aggressively between technology nodes. To analyze the impact, we separate the PE area into logic and SRAM components, and present the breakdown in Table VII for our configurations. We observe that SRAM accounts for only 9-10.6% of the total area. To assess the impact of SRAM scaling on our comparison with Gypsophila (79.80 mm^2 at 12nm), we apply 0.25× to the logic share and vary the SRAM scaling factor between 0.25× and 0.50×. Table VIII shows the results for the 20-PE configuration. Even under the most conservative SRAM scaling assumption (0.50×), our design maintains a 2.06× area advantage over Gypsophila. This is because SRAM constitutes only 9% of the total design area, limiting the impact of the SRAM scaling on our performance.

TABLE VIII
SENSITIVITY OF AREA ADVANTAGE VS. GYPSOPHILA UNDER DIFFERENT SRAM SCALING FACTORS (20 PE, 28NM SCALED TO 12NM)

SRAM scaling factor	Scaled SRAM area	Scaled logic area (0.25×)	Total area	Our advantage
0.25× (original)	3.20 mm^2	32.34 mm^2	35.55 mm^2	2.24×
0.35× (moderate)	4.49 mm^2	32.34 mm^2	36.83 mm^2	2.17×
0.40× (conservative)	5.13 mm^2	32.34 mm^2	37.47 mm^2	2.13×
0.50× (very conservative)	6.41 mm^2	32.34 mm^2	38.75 mm^2	2.06×

E. Recent FPGA Works with Multi-PE

Falic [39] presents an FPGA-based MSM accelerator that deploys multiple lightweight cores, each integrating a pipelined modular multiplier and local BRAM-based cache. The design applies batching and reuse across scalar windows and considers SLR-crossing to address intra-FPGA routing limitations. While implemented for FPGA, this SLR-aware decomposition is conceptually analogous to chiplet-to-chiplet partitioning in ASICs. In contrast, our architecture targets ASIC platforms and is optimized for physical scaling across chiplets. Rather than instantiating multiple independent MSM pipelines, we adopt a collaborative MSM execution model using mixed-window decomposition across PEs. A key design goal is to minimize on-chip SRAM footprint, which dominates area and power consumption in ASICs - unlike FPGAs, where BRAM is abundant and low-cost. While Falic reports an improvement of up to 3.9× over prior FPGA designs, no direct performance or latency numbers are reported and no comparative baselines are included. As such, a quantitative comparison remains infeasible. Instead, we want to highlight that our work focuses efficient and low memory usag, optimized placement, on chiplet-level throughput scaling.

VI. DISCUSSION

From a design tradeoff perspective, our results highlight the importance of memory-aware partitioning in ASIC-based MSM accelerators. In contrast to FPGA designs where BRAM is abundant and often underutilized, ASIC implementations must balance multiplier count, memory size, and interconnect cost carefully. Our use of full PADD units increases arithmetic density, but is justified by minimizing the number of passes and associated memory fetches. Furthermore, the mixed-window strategy not only reduces peak memory per PE but also facilitates even workload distribution across chiplets. We believe future work can explore dynamic window rebalancing and more fine-grained workload distribution to further improve chiplet-level utilization under variable workloads.

A. Thermal Management Considerations

Our two-chiplet configuration dissipates 92.2 W in total, whereas each chiplet consumes roughly 46 W over 80 mm^2. This results in a power density of about 0.575 W/mm^2, which is well within the range of commercially deployed products [32], [33], [40], [41]. In our design, the compute chiplets and HBM stacks are placed *side-by-side* on the interposer rather than vertically stacked, which limits direct thermal coupling. REED [21], CiFHEr [20], and Chiplever [22] adopt similar 2.5D integration strategies and scope their thermal analysis at the architectural modeling level. We follow the same approach and leave a detailed thermal simulation for future work.

B. Cross-Platform Comparison Considerations

The speedups reported in Section V-C span GPU, FPGA, and ASIC implementations and should be read with the platform context of Table V and Table VI in mind. Since technology node, clock frequency, and area budget differ across platforms, no single normalized metric captures the relative merits of these designs. We therefore follow the methodology of prior cryptographic acceleration work [3], [16]–[18], [35]: results are grouped by platform class, implementation parameters are reported alongside each figure, and all designs are evaluated on the same workload range, $N = 2^{16}$ to 2^{26}.

C. System-level contributions

Recent work studied chiplet-based solutions for FHE and single-die MSM designs, however, no prior work has investigated a chiplet-based MSM architecture. To the best of our knowledge, this is the first work to explore collaborative multi-PE MSM execution on chiplets. All PEs jointly compute a single MSM instance. This differs from Gypsophila [17] where each PE independently computes a separate MSM. Furthermore, our memory-aware mixed window sizing and chiplet mapping strategies are specifically designed for MSM's data flow and do not directly apply to FHE designs such as CiFHEr [20] or REED [21].

VII. 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.

Beyond raw performance gains, our approach demonstrates that chiplet-based decomposition is a viable path forward for high-throughput ZKP acceleration under ASIC constraints. The techniques presented here can be generalized to other elliptic curve workloads. Our findings suggest that future ASICs may benefit from co-optimizing memory layout and chiplet packaging jointly, rather than treating memory and interconnect as post-synthesis concerns.

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] A. Chiesa, Y. Hu, M. Maller, P. Mishra, P. Vesely, and N. Ward, "Marlin: Preprocessing zksnarks with universal and updatable srs," *Cryptology ePrint Archive*, 2019, report 2019/1047.

[3] 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.

[4] 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.

[5] 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

[6] 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

[7] 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

[8] G. Gutoski, "Multi-scalar multiplication: State of the art and new ideas," 2023. [Online]. Available: https://www.slideshare.net/GusGutoski/multiscalar-multiplication-state-of-the-art-and-new-ideas

[9] 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

[10] 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.

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

[12] 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.

[13] X. Pottier, T. de Ruijter, J. Bertels, W. Legiest, M. V. Beirendonck, and I. Verbauwhede, "OPTIMSM: FPGA hardware accelerator for zero-knowledge MSM," Cryptology ePrint Archive, Report 2024/1827, 2024.

[14] 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.

[15] 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.

[16] 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.

[17] 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
[18] 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
[19] 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
[20] S. Kim, J. Kim, J. Choi, and J. H. Ahn, "Cifher: A chiplet-based FHE accelerator with a resizable structure," in *International Symposium on Secure and Private Execution Environment Design, SEED 2024, Orlando, FL, USA, May 16-17, 2024*. IEEE, 2024, pp. 119–130. [Online]. Available: https://doi.org/10.1109/SEED61283.2024.00022
[21] A. Aikata, A. C. Mert, S. Kwon, M. Deryabin, and S. S. Roy, "REED: chiplet-based accelerator for fully homomorphic encryption," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2025, no. 2, pp. 163–208, 2025. [Online]. Available: https://doi.org/10.46586/tches.v2025.i2.163-208
[22] 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
[23] D. Bernstein, "Explicit-formulas database," 2024. [Online]. Available: https://cir.nii.ac.jp/crid/1570572699314327808
[24] 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
[25] H. Hisil, K. K.-H. Wong, G. Carter, and E. Dawson, "Twisted edwards curves revisited," Cryptology ePrint Archive, Paper 2008/522, 2008. [Online]. Available: https://eprint.iacr.org/2008/522
[26] "Chiplet design automation: Methodologies, advances, and directions," *ACM Transactions on Design Automation of Electronic Systems*, vol. 31, no. 5, 2026.
[27] 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
[28] J. Kim, J.-H. Kim, and S.-M. Lim, "Silicon interposer-based 2.5d ics for high-performance computing," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 7, no. 12, pp. 1993–2004, 2017.
[29] 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/
[30] D. D. Sharma, "Universal chiplet interconnect expresstm (ucie): Building an open ecosystem of chiplets for on-package innovations," 2023.
[31] D. H.-M. Eric Guthmuller, Jérome Fereyre, "Gpgpus on fpgas: A competitive approach for scientific computing ?" *DATE 2025 - Design, Automation and Test in Europe Conference*, March 2025. [Online]. Available: https://cea.hal.science/cea-05043041v1
[32] AMD/Xilinx, "Varium c1100 compute adaptor data sheet (ds1003)," 2013, accessed: 2026. [Online]. Available: https://docs.amd.com/v/u/en-US/ds1003-varium-c1100
[33] ——, "Alveo u280 data center accelerator card data sheet (ds963)," 2023, accessed: 2026. [Online]. Available: https://docs.amd.com/r/en-US/ds963-u280/Summary
[34] 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.
[35] 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.
[36] Rambus, "Hbm3 memory: Break through to greater bandwidth." 2023.
[37] 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
[38] 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.
[39] Y. Yang, Z. Lu, J. Zeng, X. Liu, X. Qian, and Z. Yu, "Falic: An fpga-based multi-scalar multiplication accelerator for zero-knowledge proof," *IEEE Transactions on Computers*, vol. 73, no. 12, pp. 2791–2804, 2024.
[40] Nvidia, "H100 tensor core GPU datasheet," 2023, accessed: 2026. [Online]. Available: https://resources.nvidia.com/en-us-hopper-architecture/nvidia-tensor-core-gpu-datasheet?ncid=no-ncid
[41] S. Singh *et al.*, "The amd epyc processor architecture," in *IEEE Hot Chips Symposium (HCS)*. IEEE, 2018.

VIII. Biography Section



Florian Hirner obtained his BSc and MSc degrees in computer science from Graz University of Technology, Austria, in 2020 and 2022, respectively. He is currently a PhD candidate at ISEC, Graz University of Technology. His research interests are hardware architecture and design aspects of PQC and ZKP.



Florian Krieger received the BSc degree in 2022 and the MSc degree in 2023 from Graz University of Technology, Austria. He is currently pursuing a PhD degree at ISEC, Graz University of Technology. His research interests are the design aspects of PQC, FHE, and ZKP.



Sujoy Sinha Roy (Senior Member, IEEE) is an Associate Professor of cryptographic engineering at ISEC, Graz University of Technology. He is a Co-Designer of "Saber," which is a finalist key encapsulation mechanism (KEM) candidate in NIST's Post-Quantum Cryptography Standardization Project. He is interested in the implementation aspects of cryptography.