Myosotis: An Efficiently Pipelined and Parameterized Multi-Scalar Multiplication Architecture via Data Sharing

Changxu Liu, Hao Zhou, Lan Yang, Zheng Wu, Patrick Dai, Yinlong Li, Shiyong Wu, Fan Yang, *Member, IEEE,*

***Abstract*—Zero-knowledge proof (ZKP) is a widely used privacy-preserving technology, where Multi-Scalar Multiplication (MSM) accounts for over 70% of the computational workload. The acceleration of MSM can enhance the overall performance of ZKP, making it a focal point of community attention. However, in practical applications involving the deployment of multiple MSM accelerators, existing designs often overlook strategies for optimizing bandwidth and area efficiency. To address this, we propose Myosotis, an efficiently pipelined and parameterized Multi-Scalar Multiplication architecture. By sharing input data and allocating cache effectively, it mitigates average transmission bandwidth in runtime. Myosotis also supports the use of multiple Point Addition (PADD) units to achieve performance gains, balancing area overhead and latency for improved area efficiency. Different parameter selection enables a trade-off between the performance, area, and bandwidth of the MSM accelerator. When benchmarking with MSM degrees between 2^{18} and 2^{26}, our proposed baseline design achieves up to 3.32× and 6.72× speedups over state-of-the-art FPGA and ASIC designs. Compared to the baseline, Myosotis with two window MSMs and one PADD unit reduces bandwidth demand by 43% while maintaining similar area and latency. On the other hand, Myosotis with three window MSMs and two PADD units decreases latency by 43% and bandwidth by 17%, with only a 9% area increase.**

***Index Terms*—Multi-Scalar Multiplication, Zero-knowledge proofs, Customed circuit design, Pipelined Computing**

I. Introduction

Zero-knowledge proof (ZKP) [1] has gained significant attention as a widely used privacy-preserving protocol. It is increasingly applied in various fields, including decentralized healthcare finance systems [2], ZK-rollups, and verifiable computation in deep learning [3], [4]. Blockchain provides a decentralized and tamper-proof distributed ledger [5]. ZKP protocols are also used in blockchain to verify transactions and ensure privacy, thereby enhancing security. We can leverage zero-knowledge proofs to enable one party (the prover) to demonstrate to another party (the verifier) that they possess certain knowledge without disclosing any valuable information.

zkSNARK (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge) [6] is one of the most commonly used zero-knowledge proof protocols. zkSNARK is characterized by its small proof size, fast verification, and non-interactive nature. However, the proof generation phase is time-consuming and poses a computational bottleneck, accounting for a significant portion of the computational load and time. Multi-Scalar Multiplication (MSM) and Number Theoretic Transform (NTT) are two critical computational operations in the proof generation phase. Among these, MSM consumes over 70% of the runtime [7]. Thus, enhancing MSM efficiency is crucial to prevent it from hindering the performance of algorithms.

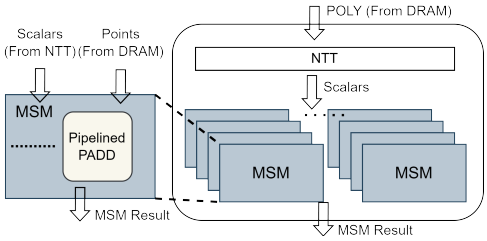

Fig. 1: A simplified accelerator schematic for ZKP proof generation phase. Multiple MSM modules and one NTT module ensure pipeline balance in the ZKP accelerator's data flow. The computational capacity of an MSM accelerator mainly comes from its pipelined PADD unit.

The Pippenger Algorithm [8], also known as the bucket method, is commonly utilized in implementing MSM. This technique enables the segmentation of MSM into smaller window MSMs, effectively harnessing its core— the pipelined point addition (PADD) units—to unleash the capability for parallel computation. Compared to the conventional double-and-add method [9], this technique significantly enhances data utilization, thereby increasing the computational efficiency of MSM. However, its computational speed may still pose bottlenecks in practical applications. Taking a simplified ZKP accelerator as an example, as shown in Fig. 1, we explore the application of MSM in real-world scenarios. Assuming an ASIC implementation with 1 GHz frequency with SAM [10] and CycloneMSM [11] as the NTT and MSM modules, respectively, the operation time of an MSM is about 24 times the operation time of an NTT for a polynomial-size of 2^{24}. We have no choice but to replicate MSM accelerators for parallel processing to achieve fully pipelined performance for enhanced efficiency. For instance, deploying 24 MSM modules alongside a single NTT module. However, this approach leads

This work was supported in part by the National Key RD Program of China under Grant 2023YFB2704600 and in part by the Science and Technology Commission of Shanghai Municipality under Project 24BC3201000. Changxu Liu, Hao Zhou, Lan Yang, Zheng Wu, and Fan Yang are with the State Key Laboratory of Integrated Chips and Systems, School of Microelectronics, Fudan University, Shanghai, China. Patrick Dai is with Semisand Chip Design Pte. Ltd., Singapore. Yinlong Li and Shiyong Wu are with Shanghai Academy of Future Internet Technology. Corresponding author: yangfan@fudan.edu.cn, wusy@safit.org.cn

to two major issues:

Bandwidth Bottleneck. As depicted in Fig. 1, both NTT and MSM modules require external POLY/Points for calculations, leading to an unacceptable total bandwidth demand of nearly 1.9 TB/s for MSM clusters, with each MSM requiring 83 GB/s of bandwidth. Moreover, this does not account for additional bandwidth for the NTT module. One potential way to optimize bandwidth requirements is by increasing window sizes, such as 16 or even 64 [11], the resulting higher area costs provide limited performance enhancements.

Low Area Efficiency. While bandwidth optimization alleviates accelerators' limitations, enhancing performance in MSM primarily hinges on increasing the number of PADD units, as depicted in Fig. 1. The standard strategy enhances the speed at the expense of area by duplicating accelerators, facilitating somewhat independent control and data flows. However, it leads to the redundant duplication of many shareable circuit components, making it an inefficient approach for area efficiency. Moreover, the utilization of PADD units tends to be relatively low.

In this paper, we introduce Myosotis, an efficiently pipelined and parameterized Multi-Scalar Multiplication architecture, based on the bucket method and the principles of data sharing. Myosotis facilitates conflict-free access for multiple requests, thereby enhancing overall pipeline performance. To reduce the transmission bandwidth requirements, Myosotis enhances data utilization by sharing points and efficiently allocating caches, thereby lowering the frequency of data input transmissions. Additionally, in pursuit of better performance and reduced unnecessary resource duplication overhead, Myosotis introduces an integrated multi-PADD design, allowing us to customize the number of window MSMs and PADD units to be accommodated. This design enhances area efficiency by leveraging shared circuit resources, enabling us to strike a trade-off between latency, area, and bandwidth.

Specifically, our contributions can be summarized as follows:

- We propose an architecture supporting parallel processing of multiple window MSMs. By sharing point data and making efficient use of caches, this design significantly reduces the average transmission bandwidth in runtime. Additionally, the architecture enhances the utilization of the PADD units, thereby improving the overall pipeline performance.
- We propose an integrated multi-PADD design that allows Myosotis to share resource logic more efficiently, resulting in improved area efficiency. Furthermore, this design provides Myosotis with enhanced flexibility to accommodate trade-offs between area and latency.

Myosotis is a parametric architecture for constructing MSM, offering more flexible parameter selections. The proposed baseline outperforms state-of-the-art FPGA and ASIC designs, achieving speedups of up to 3.32× and 6.72×, respectively. Furthermore, in comparison to the baseline, Myosotis, with two window MSMs and one PADD unit, reduces bandwidth demand by 43% while maintaining similar area and latency. Also, with three window MSMs and two PADD units, it decreases latency by 43% and bandwidth by 17%, with only a 9% area increase.

II. Related Works

Several works have already developed high-performance accelerators for MSM based on the bucket method. PipeZK [12] is an early ASIC-based accelerator for zk-SNARK, which for the first time proposed a custom circuit solution based on the bucket method. However, its conservative architecture design limits scalability to larger window sizes and imposes high bandwidth requirements, limiting its performance. Gypsophila [13] is a scalable and bandwidth-optimized architecture for multiple MSM tasks. It achieves streamlined data flow through algorithmic decomposition of the bucket method and improves throughput by sharing data and resources. However, it primarily focuses on pipelining optimizations for the hardware implementation of the bucket method, offering limited hardware optimization for single MSM tasks. Despite Gypsophila's efforts to minimize the access conflict properties of its bucket structure, point pairing collisions still occur in its data flow, impacting performance. Additionally, its scalability depends on the increase in MSM PEs, with the number of supported window MSMs being equal to the number of PADD units. This setup lacks flexibility and does not achieve an optimal balance between area overhead and latency. MSMAC [14] introduces a specially designed ISA in the MSM acceleration architecture, highlighting the batching of point addition operations. Its runtime system can divide MSM tasks into multiple subtasks, which are then processed separately in different PEs. However, the specific ISA is overly simplistic, focusing more on high-level task scheduling optimizations while neglecting improvements in underlying hardware efficiency. ReZK [15] is a recently released ZKP accelerator built using reconfigurable technology, which also incorporates acceleration for MSM. However, we believe that reconfigurable technology is not well-suited to optimizing the computational characteristics of MSM, and the performance improvements it achieves are rather limited. PriorMSM [16] explores the design space of key parameters in MSM accelerator design, but its focus is restricted to optimizing single-engine MSM accelerators. Additionally, its optimization objectives are limited to area and execution time, without considering bandwidth requirements or the construction of multi-engine MSM accelerators.

ZodiacMSM [17] is optimized for multi-chip integration computing MSMs, featuring memory access partitioning for improved performance and scalability. PipeMSM [18], CycloneMSM [11], HARDCAML [19], and SuperScalar [20] are all FPGA-based MSM accelerators that optimize the implementation from an engineering perspective, exhibiting impressive performance. However, they still suffer from data dependencies in the data flow, resulting in pipeline bubbles. BSTMSM [21] is another FPGA-based MSM accelerator that proposes a Barrel State Tracking method to reduce bucket collisions. However, its implementation involves extensive use of caches to store intermediate data, which, while improving computational efficiency, is considered to have low area efficiency.

cuZK [7] and GZKP [22] propose GPU-based ZKP accelerators that leverage on-chip cache and computational resources of GPUs to maximize efficiency through the bucket method and precomputation techniques, albeit with higher memory overhead. Elastic MSM [23] focuses on enhancing the parallel Pippenger Algorithm on GPUs through precomputation techniques, exploring a trade-off between storage overhead and running time. However, using GPUs to accelerate based-finite field operations can lead to low utilization of the GPU's resources that are primarily designed for floating-point operations. Additionally, their design approach is fundamentally different from that of custom hardware.

In summary, we can identify the limitations of previous works, which mainly focus on the following points:

- Their setups typically involve a single window MSM and a single PADD. While some related works support multi-window MSM and multi-PADD configurations, they often do not allow for flexible, independent settings of the number of window MSMs and PADDs, essentially replicating multiple sets of window MSMs and PADDs.
- Most previous works neglect the efficient utilization of data, which is crucial for saving interface bandwidth and reducing the area overhead of shared circuit components.
- Previous works still encounter issues with data dependencies in the pipeline and conflicts when accessing the bucket, which diminishes the utilization of PADD modules and computational efficiency. The FIFOs containing conflicting point pairs may repeatedly reinsert conflicted points into the original queue, potentially increasing the frequency of interrupting burst transfers of external data, thereby weakening performance.
- Most previous works have ignored discussions on hardware parameter configurations for MSM accelerators, thus overlooking the design space exploration for the architectures. Consequently, the area efficiency of these accelerators may not be optimal.

Our proposed design addresses the limitations of the previous works by supporting conflict-free access for multiple requests and improving data utilization through sharing. This design achieves high-performance computing with an efficient pipelined implementation, while also considering interface bandwidth reduction. As an accelerator architecture, reducing bandwidth usage also helps lower costs.

III. Preliminary

A. Elliptic Curves

The data for Multi-Scalar Multiplication consists of scalars and points on elliptic curves. Considering security concerns, computations involving points on elliptic curves are performed within finite fields, introducing substantial complexity. Mathematically, the points on an elliptic curve E over the field F_q satisfy Eq. 1, with the condition that $4a_{sw}^3 + 27b_{sw}^2 \neq 0$. Eq. 1 is also known as the Short Weierstrass form of an elliptic curve.

$$y^2 = x^3 + a_{sw} \cdot x + b_{sw}. \tag{1}$$

In addition to the Short Weierstrass curves, elliptic curves can also be expressed in other forms, such as Montgomery curves, as shown in Eq. 2, and Twisted Edwards curves, as shown in Eq. 3. When specific conditions are met, mathematical mappings can be applied to transform one curve form into another. Additionally, the points on the curve also undergo mathematical transformations [24].

$$B_{mon} \cdot y^2 = x^3 + A_{mon} \cdot x^2 + x. \tag{2}$$

$$a_{te} \cdot x^2 + y^2 = 1 + d_{te} \cdot x^2 \cdot y^2. \tag{3}$$

Any curve can be transformed into the Short Weierstrass curve. Additionally, each Twisted Edwards curve is birationally equivalent to a Montgomery curve over the field K. However, transforming an elliptic curve in the Short Weierstrass form over the base field F to the Montgomery form requires satisfying specific mathematical conditions [25], [26]. Due to variations in computational complexity for point addition and point double across different curve forms, the property of birational equivalence between different curve forms allows us to streamline calculations on certain elliptic curves.

Twisted Edwards curves excel in computational efficiency, thanks to their optimized addition formula. This advantage stems from the robust completeness property of the curve, leading to improved performance of cryptographic protocols. For instance, the BLS12-377 curve is a widely used curve form. Its mathematical expression in Short Weierstrass form is shown in Eq. 4.

$$y^2 = x^3 + 1. \tag{4}$$

However, it can also be transformed into a Twisted Edwards curve with $a = -1$. Due to the property of $a = -1$, point addition and point double on this curve have a lower computational complexity. Through convenient pre-processing, we achieve reduced computational complexity of point operations.

In addition to considering the form of an elliptic curve, it is crucial to select an appropriate coordinate system to improve the efficiency and security of point operations. In the given equations, affine coordinates (x, y) is used. Under the affine coordinate system, the unified point addition law requires 7 modular multiplications, 2 modular inversions, and 4 modular additions. Inefficient modular inversions can hinder the implementation of point addition in a pipelined manner, thereby reducing overall performance.

The use of projective coordinates (X, Y, Z) eliminates the need for modular inversions in point addition. Affine coordinates (x, y) are equivalent to $(X/Z, Y/Z)$. Furthermore, we highly recommend using extended projective coordinates (X, Y, Z, T) by introducing an auxiliary coordinate $T = XY/Z$. The unified point addition law based on extended projective coordinates offers lower computational complexity than the one based on projective coordinates. Many relevant works have adopted extended projective coordinates for constructing the PADD unit [11], [21], which aligns with the coordinate system selected in our paper.

B. Pippenger Algorithm

The MSM represents the outcome of point addition (PADD) resulting from multiple scalar multiplications (SM), which is also recognized as point multiplication (PMUL). This process

can be further dissected into a sequence of point addition (PADD) and point double (PDBL) operations. The mathematical expression for MSM is represented by Eq. 5, where a_i represents an integer scalar, G_i denotes a point on the elliptic curve, and the variable N signifies the degree of MSM. PDBL and PADD operations on elliptic curves exhibit complexities significantly higher than standard arithmetic addition operations.

$$MSM(\vec{a}, \mathbb{G}) = \sum_{i=0}^{N-1} a_i \cdot G_i. \tag{5}$$

For problems that involve only a finite number of SMs, such as in EdDSA [27] and other digital signature schemes, the double-and-add method is commonly used. Some approaches also incorporate precomputation techniques to improve the computational efficiency of SM at the expense of increased storage overhead [28]. Within ZKP challenges, the MSM algorithm typically involves a large number of SMs, implying a higher degree. With the ever-increasing computational demands of contemporary applications, N has the potential to scale up to 2^{20} or even 2^{30}. The Pippenger Algorithm can markedly optimize the computational performance of MSM. Fig. 2 presents the process of the Pippenger Algorithm. Here, the bit width of the scalar a_i is b. Each scalar a_i is segmented into $m = \lceil \frac{b}{c} \rceil$ smaller scalars, indexes, of c-bit each, represented as $a_i = \text{concat}\{a_{ij}\}$. We refer to the PMUL of each set of indexes and their corresponding points as window MSM, symbolized as R_j.

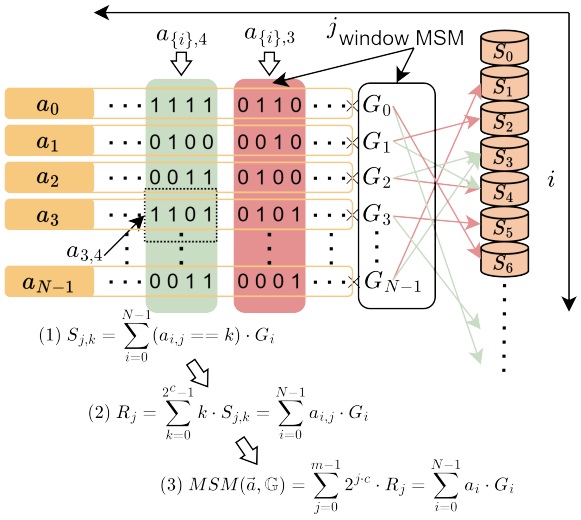

Fig. 2: The process of the Pippenger Algorithm. In this case, the window size c is set to 4. The algorithm utilizes $S_{j,k}$ to represent the bucket's number, where different window MSMs have different j for their buckets. However, for the sake of simplicity in the illustration, we use S_k to represent the buckets. Different window MSMs reuse the same set of buckets.

The Pippenger Algorithm can be divided into three steps: Bucket Classification, Bucket Aggregation, and Window Aggregation. In the Bucket Classification step, the points are assigned to their corresponding buckets based on the correspondence between index $a_{i,j}$ and bucket number k. As a result, we obtain S_k, the set of points in each bucket. This step involves a total of $m \cdot (N - 2^c)$ PADD operations, as shown in (1) of Fig. 2. In the Bucket Aggregation step, the points from 2^c buckets are aggregated to obtain the results of window MSM. This step requires approximately $m \cdot 2^c$ PADD operations, corresponding to (2) in Fig. 2. In the Window Aggregation step, the results of m window MSMs are aggregated to obtain $MSM(\vec{a}, \mathbb{G})$. This step consumes $m - 1$ PADD operations and $c \cdot (m - 1)$ PDBL operations, as shown in (3) of Fig. 2. Bucket Classification is closely correlated with the degree. The computational complexity of the Bucket Aggregation step and Window Aggregation step is solely determined by the window size and the scalar width, which is much lower than that of the Bucket Classification step. For instance, when the degree of MSM is 2^{20} and the window size is 12, the PADD operations of the Bucket Classification step account for 99.8% of the total PADD operations. Several works have optimized the Bucket Classification step by developing dedicated circuits, while the processing of other steps is carried out on the host [11], [18], [19].

C. zkSNARK

We present the basic workflow of zkSNARK, which mainly comprises three major stages: Setup, Prove, and Verify. The prover needs to convince the verifier that, for a given program F and input x, the prover knows a witness w such that $F(x, w) = 0$. In the Setup phase, the system generates many random parameters, including the proving key and verification key. The proof system also generates vector sets based on the input, which are used in the proving phase. Each vector in the vector sets consists of N large integers within a finite field F_r, typically ranging from 256-bit to 768-bit in bit width. The proving key comprises two sets of elliptic curve points. During the proving process, the vectors are first processed through NTT/INTT and polynomial operations to obtain $H(x)$. This is followed by applying INTT to get the coefficient representation $\vec{h}$ of $H(x)$, which constitutes the POLY step. The second part is the MSM step, where $\vec{h}$ from POLY and a vector $\vec{s}$ are combined with the proving key to perform MSM operations. The degree of MSM is also N, and the results are two points on the elliptic curve. These points form the proof Π. In the verification phase, the verifier uses the verification key and the input x to check if the proof generated by the prover meets the requirements, thereby confirming whether the prover truly knows the input x.

IV. Myosotis

Myosotis is parameterized, allowing independent configuration of the number of window MSMs and PADDs, as long as the former exceeds the latter. The number of window MSMs impacts the transmission bandwidth requirements, while the number of PADD units affects performance. This flexibility enables us to efficiently explore different hardware configurations to find the most optimal implementation. This is particularly practical for future physical implementations, as we always aim to strike a balance between resource overhead and performance. Fig. 4 provides a comprehensive overview of

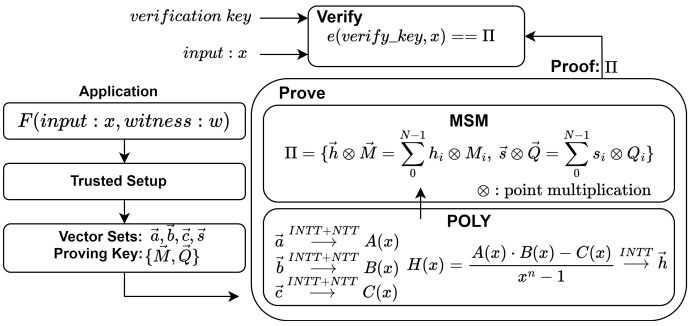

Fig. 3: The proof generation phase of zkSNARK. It mainly consists of two steps: MSM and POLY, primarily involving polynomial operations, NTT/INTT, and MSM.

the distinctions between our proposal and the previous design. Overall, Myosotis has several advantages as follows:

- Points can be shared among different window MSMs, maximizing their utilization and effectively reducing the overall transmission bandwidth requirement.
- The sharing of points brings the added benefit of higher utilization of PADD units.
- Our proposal offers greater flexibility in balancing area and performance trade-offs while optimizing bandwidth requirements, as it allows for the specification of the number of window MSMs and PADD units.

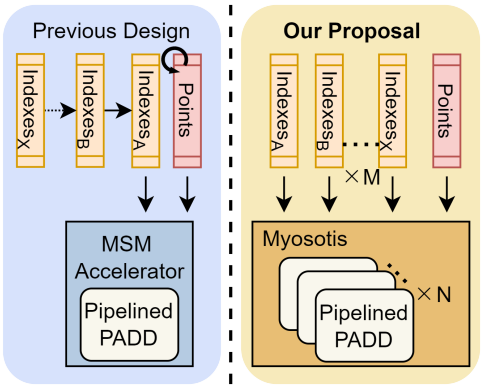

Fig. 4: Comparison of previous design and our proposal. Myosotis supports the flexibility of setting the number of window MSMs and the number of PADD units to be processed in parallel to improve bandwidth and performance, as long as the former is greater than the latter. The previous design can only be replicated as a whole multiple times or copied in a time-sharing manner.

A. Overall Architecture

The overall architecture of Myosotis is depicted in Fig. 5, which is primarily divided into four parts: Scheduler, Bucket Array, FIFO Array, and PADD Array. The Scheduler is responsible for receiving incoming data from external sources and intermediate data generated by the PADD Array. It preprocesses the data, dispatches appropriate requests to the Bucket Array and FIFO Array, and ensures timing alignment for accessing the Bucket Array. The Bucket Array comprises Bucket Banks and their corresponding Score Boards. Buckets store intermediate results of the Bucket Classification step and are dynamically updated based on the progress of computation. These buckets are constructed using true dual-port SRAMs, with their capacity dependent on the number of window MSMs and the window size. The Score Board records the validity of data in buckets using single-bit registers. Given that Buckets Array is essentially a storage unit, we argue that the register-based construction method, as adopted by PipeZK [12], incurs higher power consumption compared to the SRAM-based approach, even when using true dual-port SRAM for bucket construction. Using our proposed baseline architecture, we construct Bucket Arrays based on both SRAM and registers and conduct a simple comparison. The results show that the MSM accelerator with a true dual-port SRAM-based Bucket Array achieves lower power consumption and higher energy efficiency compared to the register-based design. The FIFO Array functions as a buffer, consolidating data outputted by the Bucket Array and forwarded by the Scheduler. We develop an Arbiter responsible for managing the FIFO Array's read logic and data output scheduling. This ensures balanced data retrieval and minimizes occurrences of external data collision. The PADD Array features an integrated multi-PADD design, providing the computational capacity of Myosotis. It retrieves data from the FIFO Array for calculation and forwards the results back to the Scheduler.

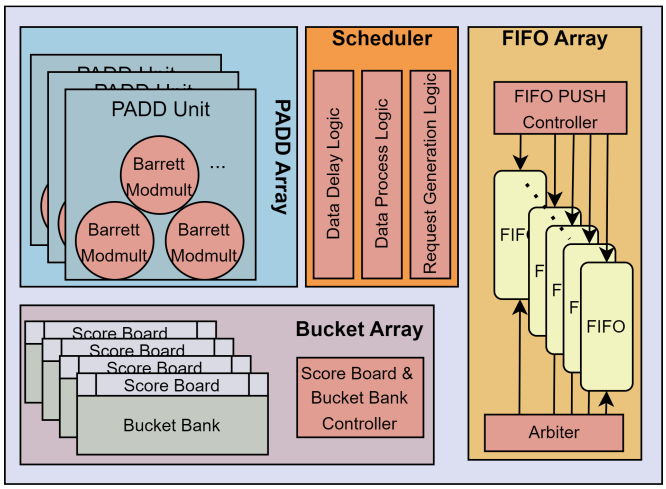

Fig. 5: Overall architecture of Myosotis. Specific quantities are not depicted in the figure, as the exact configuration is determined by specific parameters.

B. Single Window MSM and Single PADD

We refer to an accelerator that processes only one window MSM at a time and is equipped with only one PADD unit as the baseline, which is the simplest abstraction in our proposed design. Most of the previous custom circuit-based MSM accelerators can be abstracted into the baseline we propose. The data flow in our baseline also mirrors the general paradigm of Myosotis, as shown in Fig. 6. Sec. IV-C and Sec. IV-D will provide detailed insights into Myosotis'

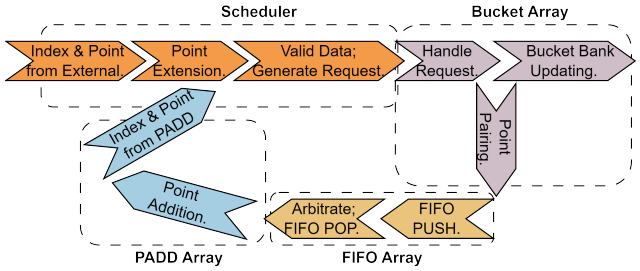

Fig. 6: The data flow of our baseline, aligns with the fundamental paradigm of data flow within Myosotis.

capability for parallel computation of multiple window MSMs and the design philosophy of integrated multi-PADD design.

In this simplest configuration of the accelerator, both the indexes and the points within the window MSM enter the Scheduler together. The Scheduler performs point extension by calculating the corresponding T coordinate in runtime based on the point's X and Y coordinates, leading to reduced bandwidth constraints. Then, the Scheduler checks the data, such as identifying whether the index is zero or negative. Subsequently, the Scheduler generates a request for the point and sends both the request and the point to the Bucket Array. The Bucket Array handles the received request, determining the point's processing based on the state queried from the Score Board, updating data recorded in the Bucket Bank, or pairing this point with the point in the Bucket Bank. Points successfully paired in the Bucket Bank are sent to the FIFO Array and pushed into the corresponding FIFO. The Arbiter in the FIFO Array manages FIFO readouts to ensure a complete and balanced distribution of data. The PADD unit within the PADD Array handles the hardware implementation of point addition algorithms. This is a fully pipelined PADD unit, guaranteeing one point addition result output per cycle and achieving extremely high throughput. The points and indexes output from the PADD are reintroduced to the Scheduler along with new indexes and points from external sources, repeating the aforementioned data flow process.

The Scheduler operates with two data streams: one handles points and indexes from external sources, while the other manages data from the PADD. The Bucket Bank within the Bucket Array needs to accommodate concurrent access to up to two requests. To prevent potential data collision caused by conflicts between two read/write requests, we implement this structure using true dual-port SRAM. Additionally, the FIFO Array includes multiple FIFOs to receive multiple point pairs that may be generated within the same cycle.

We use extended projective coordinates to represent points on Twisted Edwards elliptic curves. The PADD unit, built on the point addition formula in [29], features 9 Barrett modular multiplication [30] units in a fully pipelined structure with a 77-clock cycle latency. During scalar division into indexes, we preprocess it to map from unsigned representation a_i to signed representation $\{a_{i,(m-1)}, \cdots, a_{i,1}, a_{i,0}\}$. Leveraging properties of points on the elliptic curve, the mathematical transformation rule from $P = (X, Y, Z, T)$ to $-P = (-X, Y, Z, -T)$, this preprocessing reduces the Bucket Bank's area cost by half.

C. Multiple Window MSMs via Data Sharing

In Myosotis, the number of window MSMs is parameterized. When this number exceeds 1, multiple window MSMs can be processed in parallel and out of order. As each iteration can accommodate more indexes, the corresponding points are shared among multiple window MSMs, thereby reducing the frequency of point transmissions and allowing the data from each transmission to be fully utilized within a longer time. Each window MSM operates independently with points and its own set of indexes, resulting in separate instruction flows, Bucket Banks, and Score Boards. However, since input points are the same, Myosotis can share the data flow among multiple window MSMs, making this design hardware-friendly and reducing some register logic. Fig. 7 illustrates the processing flow using a simple case, Myosotis with two window MSMs and one PADD unit, where 0 to 3 point pairs can be generated per cycle. The Bucket Array processes requests from window MSMs based on Index_A and Index_B, and PADD computation results, simultaneously. Requests are generated independently and processed in a pipelined fashion. Point pairs are pushed into the FIFO Array when the requests arrive. These two window MSMs share the computational capacity of Myosotis, creating an impression of being processed in parallel.

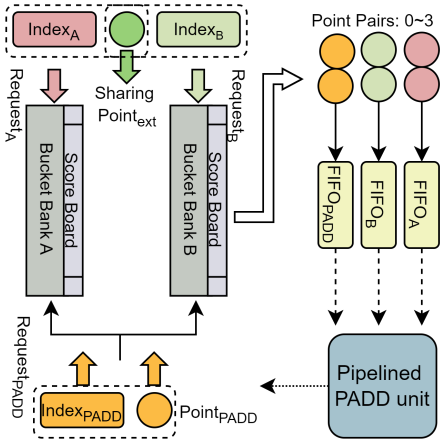

Fig. 7: Myosotis with two window MSMs and one PADD unit. Index_A and Index_B belong to the same scalar but under different window MSMs, while Index_PADD denotes the index returned by PADD.

The design principles also improve PADD utilization, as illustrated in Fig. 8. Processing more window MSMs in parallel raises the point pairing rate, thereby reducing pipeline bubbles during PADD's active time. We can conduct a simple analysis in theory. Ideally, given that each point has a pairing success probability of 1/2, the baseline appears to perfectly generate one pairing point per clock cycle. However, it's important to note that at this point, the probability of successful pairing is only theoretically equal to the efficiency of PADD's read operation. If points fail to pair, idle PADD will introduce bubbles to the pipeline. In our case, theoretically, 1.5 pairing points are obtained per cycle. The increased likelihood of

successful point pairing surpasses PADD's read operation efficiency, reducing pairing failures and enhancing PADD utilization. However, due to pipeline startup and drain times, PADD's pipeline efficiency will only approach but not reach 100%. Fig. 9 shows statistics on pipeline bubbles for three single-PADD cases, indicating that Myosotis enhances PADD pipeline utilization.

During Bucket Aggregation and Window Aggregation steps, multiple Bucket banks structure allows data from multiple window MSMs to be processed in a pipelined manner, enhancing computational efficiency in these two steps.

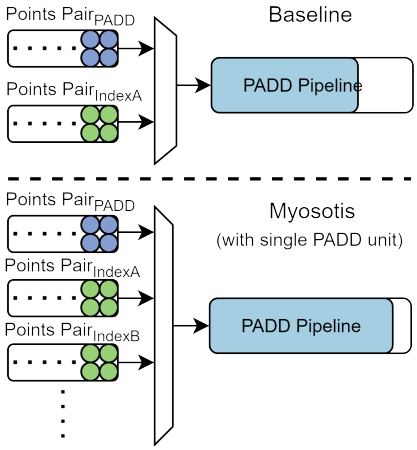

Fig. 8: In Myosotis, the computational capacity of the PADD unit is more effectively utilized compared to that in the baseline due to higher pairing efficiency.

The Myosotis's bandwidth reduction results from efficiently utilizing bucket capacity, which determines the maximum window size and the number of window MSMs processed in parallel. With the same cache size limit, Myosotis can effectively utilize cache by reducing the capacity of individual Bucket Banks and employing multiple Bucket Banks to accommodate intermediate results of the Bucket Classification step for a greater number of window MSMs. This may cause slight performance loss, but it's minor compared to the significant bandwidth reduction. However, both approaches have similar area overheads, implying a break in bandwidth bottleneck allows more accelerators per chip. Such an approach also facilitates the expansion of computational units, thereby increasing computational efficiency. Details are provided in Sec. IV-D. For example, using the same SRAM capable of holding 8192 points, the performance loss of Myosotis with two window MSMs and one PADD unit is approximately 5% compared to the baseline while reducing bandwidth demand by 50%. By enhancing PADD utilization, Myosotis alleviates performance degradation from reduced single Bucket Bank capacity. Sometimes, we are open to accepting a minor increase in chip area, by adding more Bucket Banks, in exchange for a significant reduction in bandwidth.

D. Integrated Multi-PADD Design

Myosotis introduces a bandwidth optimization strategy in Sec. IV-C. Building on this, Myosotis facilitates the utilization

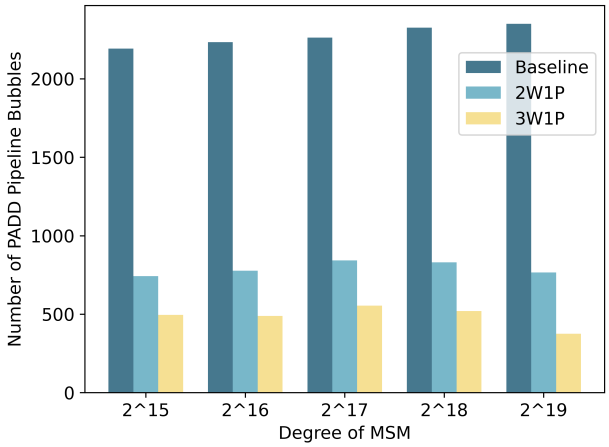

Fig. 9: The number of pipeline bubbles in three typical single-PADD cases with single window MSM (Baseline), two window MSMs (2W1P), and three window MSMs (3W1P). A lower value is preferable.

of multiple PADDs, as the computational performance is now bounded by the computing capacity of PADD units. In Sec. IV-C, we describe Myosotis's bandwidth optimization strategy, which is achieved through the allocation of bucket capacity. This architecture, based on a multi-bank bucket structure, significantly reduces the demand for interface bandwidth with minimal performance loss. Additionally, it supports handling more requests due to the increased number of read/write ports. This enables our architecture to support multiple PADD units, which generate more intermediate results and, consequently, more requests to Bucket Array. Since a single Bucket Bank struggles to effectively handle multiple requests generated by multiple PADD units, this overflow of requests can significantly hamper performance.

Myosotis supports multiple PADD units, which can significantly enhance MSM computational performance. Ideally, performance should increase linearly with the number of PADD units. However, without proper scheduling, request overflow in a single Bucket Bank can occur, leading to underutilization of PADD units. To address this, we propose an integrated multi-PADD design. This design not only incorporates more PADD units but also introduces specialized instruction and conflict-free arbitration logic to manage read and write operations in the FIFO Array. The workflow of the integrated multi-PADD design is shown in Fig. 10. In Fig. 10 (a), the Bucket Array and Scheduler provide the FIFO Array with points pairs, each associated with an instruction. The instruction format, shown in Fig. 10 (c), includes the FIFO index and the point pair's address in the Bucket Bank. This information ensures that point pairs are written to the correct FIFO, with the subscript of the FIFO Group indicating the source of the point pair. The customized arbiter reads data from the FIFO Groups in a balanced manner and sends it to the PADD Array, allowing multiple PADD units to perform computations in parallel. Fig. 10 (b) illustrates the arbitration logic of the customized arbiter. Since there will be more FIFOs than PADDs, the system needs to accommodate a sufficient number of potential point

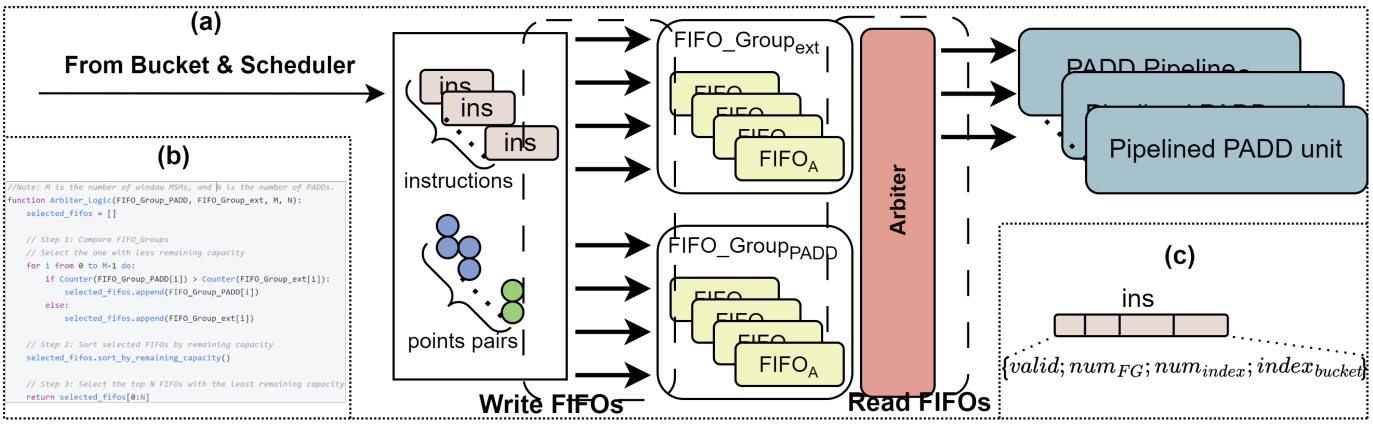

Fig. 10: (a) Integrated Multi-PADD Design. Each point pair corresponds to an instruction containing information about its destination, aiding in the FIFO write logic. The arbiter uses conflict-free arbitration logic to determine the FIFO read order, sending data to the PADD array for computation. Together, they form the integrated multi-PADD design. (b) Arbitration logic. It selects the appropriate FIFO for reading based on the remaining available capacity to balance the data flow and reduce congestion. (c) Instruction Format. The instruction includes a signal indicating whether the current computation point pair is valid, the index of FIFO_Group, the index of the window MSM to which the point belongs, and the index value indicating the bucket address.

pairs. The arbiter balances FIFO capacity by selecting the FIFO with the least remaining capacity, minimizing pipeline stalls caused by the full status of FIFOs, which can impair performance. The FIFO selection is not arbitrary, as multiple FIFOs may qualify.

Incorrect arbitration logic can result in multiple PADD units reading the same point pair within a window MSM per cycle, leading to request overflow for the associated Bucket Bank and causing performance bottlenecks. The customized arbiter balances the remaining capacity of FIFOs in the FIFO Groups, fully leveraging parallel computing capabilities. By adjusting the number of PADD units and the corresponding FIFO units, the design can be tailored to specific needs. This adjustment also increases the number of point pair instructions, and the arbiter's arbitration logic is scalable to accommodate these changes. Adding more PADD units increases computational throughput with minimal additional area, enabling shared use of components like the Scheduler, FIFO Array, and Bucket Array, thus enhancing area efficiency and computational density.

Integrated Multi-PADD Design can enhance the computational performance and area efficiency of Myosotis, but they may also undermine the bandwidth optimization effect due to the overall increase in throughput. Myosotis supports flexible configuration of the number of PADD units to explore the optimal balance between performance, area, and bandwidth requirements in practical applications. We advocate for a principle in Myosotis: **the number of window MSMs should surpass the number of PADD units**. When there are more PADDs than window MSMs, the probability of successful point pairing is lower than the computational efficiency of PADDs, further reducing their utilization. Given that PADDs are costly, it's essential to maximize their utilization. For instance, in a scenario with two window MSMs and three PADD units, ideally, each cycle would produce (2+3) * 1/2 point pairs. However, since the PADD Array can read three point pairs per cycle, it may introduce more bubbles into the PADD pipeline. This imbalance, where there are more PADD units than window MSMs, leads to a more noticeable reduction in PADD utilization. This imbalance can also result in an overflow of requests, where requests are generated at a faster speed than they are consumed, limiting the overall performance to the point pairing stage. Another key factor is bandwidth: if there are more PADD units than window MSMs, the bandwidth demand exceeds that of the baseline by approximately $\frac{\text{Num.}_{\text{PADD}} - \text{Num.}_{\text{win}}}{\text{Num.}_{\text{win}}}$, ultimately constraining overall performance due to bandwidth limitations.

V. Results

In this section, we discuss the evaluation results of Myosotis, a parameterized architecture developed in Verilog HDL. The experimental setup, including various representative parameter combinations, is detailed in Tab. I, with all synthesized accelerators operating at a frequency of 1 GHz. Our baseline configuration aligns with most existing MSM accelerators [11], [12], [18], [19], [21]. For Myosotis with more parameter configurations, we select a typical range for the number of window MSMs and PADD units. This range is consistent with the scale of the latest and larger MSM accelerators [14], [20], [31], making it a practical parameter choice. Additionally, Myosotis under various parameter configurations can generally be abstracted into these selected typical cases. Although Myosotis is highly scalable, we still do not recommend setting an excessive number of window MSMs. Too many window MSMs can lead to increased computation workload and larger intermediate data storage, potentially degrading performance. This can also present greater challenges for physical implementation. We use the BLS12-377 as the target curve because it is commonly used in practical applications. However, the

proposed design is general and modular, making it adaptable to other types of curves as well.

TABLE I: Experimental Configuration.

Platform		TSMC 12nm process
Synthesis Tool		Synopsys Design Compiler
Frequency		1 GHz
Target Curve		BLS12-377
Benchmark	Baseline	1 window MSM 1 PADD
	Myosotis	2 window MSM 1 PADD
		3 window MSM 1 PADD
		3 window MSM 2 PADD
		4 window MSM 3 PADD

To the best of our knowledge, previous works can only be abstracted into our proposed baseline. Therefore, we compare the proposed baseline with previous works and also contrast the baseline with other typical cases of Myosotis to explore the trade-offs in area, latency, and bandwidth in our work.

The choice of Window Size: the scale of window size affects two factors:

- Area Overhead of the Bucket.
- Computational Workload.

We believe that the area overhead of the Bucket Bank should not be excessive, as the performance gains from increasing the depth of a single Bucket Bank often diminish. From the perspective of minimizing computational load, Work [14] recommends a window size of 13, while work [18] suggests a window size of 12. We also found that, when the window size is 12 or 13, the area of the Bucket Bank and the area of a PADD unit are balanced in the TSMC 12nm process, contributing to optimal area efficiency. Therefore, we set the window size to 13 in our proposed baseline.

A. Evaluation of Proposed Baseline

We compare our proposed baseline design in terms of both the number of clock cycles and actual execution time. The former is to ensure a fair comparison without considering differences in operating frequency, while the latter aims to provide a more intuitive illustration of the speedup of our design relative to previous works.

In terms of the number of clock cycles, we compare our proposed baseline with BSTSM and PipeZK, which are representative works implemented on FPGA and ASIC, respectively. Fig. 11 (a) uses an MSM with a degree of 2^{20} as the benchmark, while Fig. 11 (b) uses an MSM with a degree of 2^{25} as the benchmark, though this data of PipeZK is not available for the latter. The proposed baseline has a clock cycle count similar to that of BSTMSM, and they remain almost consistent as the MSM degree increases. However, We observe the difference in their cache capacities, with BSTMSM boasting a larger cache of 15MB, far surpassing the proposed baseline's limited 768KB. We believe this is because BSTMSM stores a large amount of intermediate data on-chip. In contrast, we choose to perform reduction computations on intermediate data as soon as possible to reduce storage overhead and improve area efficiency. As a result, our speed remains almost at the same level as BSTMSM. Notably, a 15MB true dual-port SRAM on the TSMC 12nm process occupies tens of square millimeters, far exceeding the area of our entire accelerator. Considering overall computation density, we believe the proposed baseline demonstrates superior comprehensive performance. Compared to PipeZK, despite PipeZK having two PADD units which theoretically should provide greater computational power, its scheduling mechanism is not smart enough and its choice of window size is overly conservative. As a result, its performance is significantly lower than our proposed baseline.

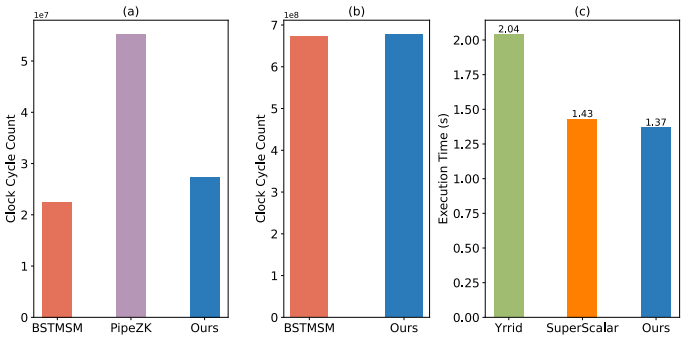

Fig. 11: (a) Comparison of Clock Cycle Counts Between Our Proposed Baseline and State-of-the-Art FPGA and ASIC Designs Using an MSM with a Degree of 2^{20} as the Benchmark. (b). Comparison of Clock Cycle Counts Between Our Proposed Baseline and State-of-the-Art FPGA Design Using an MSM with a Degree of 2^{25} as the Benchmark. (c) Comparison of latency with the latest designs in ZPrize 2023, benchmarked with batch-4 MSM with degree 2^{24}. It's important to note that these two works deploy three EC Adders (equivalent to the PADD unit in our design) on-chip, whereas our proposed baseline only includes a single PADD unit.

In terms of the actual implementation time, Tab. II presents a comparison between our proposed baseline and previous works. The comparison is conducted with FPGA-based works that have a window size comparable to ours. When it comes to ASIC-based designs, PipeZK represents the state-of-the-art implementation at now. It can be observed that the proposed baseline achieves up to a 3.32 $\times$ speedup relative to the best FPGA-based design, BSTMSM. Compared to PipeZK, our proposed baseline achieves a speedup of up to 6.72 $\times$. However, comparing designs with different frequencies is not straightforward.

As the degree of MSM increases, the speedup tends to rise. This trend occurs because when the MSM degree is small, the time consumed by the Bucket Aggregation step and Window Aggregation step is relatively high, especially when the gap between the MSM degree and the window size is small. However, the proportion of time spent on these two steps decreases as the degree of MSM increases, as they are independent of the MSM degree and only depend on the window size. One feasible mitigation strategy is to increase the 'N' per turn to reduce the number of turns and enhance computational performance for smaller MSM degrees. While Elastic MSM [23] offers a solution by using the precomputation technique to fold the turns and increase the 'N' per turn, it also significantly increases storage overhead.

TABLE II: Comparison of Proposed Baseline with Typical FPGA/ASIC Implementations.

Design		BSTMSM [21]	PipeMSM [18]	HardCaml [19]	PipeZK[a] [12]	**Baseline**[b]
Platform		Xilinx U250	Xilinx U55C	Xilinx VU9P	UMC 28nm	TSMC 12nm
Curve		BLS12-377	BLS12-377	BLS12-377	BLS12-381	BLS12-377
Frequency		300 MHz	125 MHz	278 MHz	300 MHz	1 GHz
Window Size		13	12	13	4	13
Bucket Capacity		15 MB	576 KB	14.7 MB	1.4 KB	768 KB
Area		411k LUTs / 744k REGs/ 2920 DSPs / 623 SRAMs	-	387k LUTs / 733k REGs 2999 DSPs / 883.5 SRAMs	33.72 mm^2	5.36 mm^2
Time (ms)	2^{18}	23	68.8	-	46	**11.7 (1.97 × / 3.93 ×)**
	2^{19}	40	136.6	499	92	**16.9 (2.37 × / 5.44 ×)**
	2^{20}	75	273	540	184	**27.4 (2.74 × / 6.72 ×)**
	2^{21}	145	-	620	-	**48.4 (3.00 × / -)**
	2^{22}	285	-	780	-	**90.3 (3.16 × / -)**
	2^{23}	564	-	1094	-	**174.2 (3.24 × / -)**
	2^{24}	1124	-	-	-	**342.0 (3.29 × / -)**
	2^{25}	2242	-	-	-	**677.6 (3.31 × / -)**
	2^{26}	4479	-	-	-	**1348.7 (3.32 × / -)**

[a] PipeZK is equipped with two PADD units on-chip, but only performs the bucket classification step on the chip, with the remaining steps executed on the host.
[b] The data in parentheses show speedup relative to other works, with the left value compared to BSTMSM and the right to PipeZK.

Myosotis addresses this by processing multiple window MSMs in parallel, increasing the 'N' per turn.

Comparasion with ZPrize 2023: We also compare our proposed baseline with two recently released FPGA-based designs in ZPrize 2023 [32], SuperScalar [20] and Yrrid [31] in Fig. 11 (c). It can be observed that our proposed baseline also demonstrates a speed advantage over these designs. It is worth noting that this is just the baseline design of our proposed architecture, Myosotis. In our view, achieving such performance already positions it as a representative of mainstream MSM accelerators, even surpassing them. In Sec. V-B, we will further explore Myosotis's performance to illustrate the superiority of our proposed architecture.

Comparison on other curves with existing works: In addition to evaluating MSM on the 377-bit BLS12-377 curve, we also compare the performance of our proposed baseline on other elliptic curves with different bit-widths, including BN128 and MNT4753. The detailed comparison data is shown in Tab. III, where the speedup of our baseline relative to other works is provided in parentheses. For the BN128 curve, it is important to note that MSMAC is an FPGA-based MSM accelerator for the BN128 curve, equipped with 4 PADD units. PipeZK also uses 4 PADD units. In contrast, our baseline only utilizes a single PADD unit. Our baseline achieves speedup of up to 1.38 × and 2.47 × relative to these two works. As the degree of MSM increases, the performance advantage of our baseline relative to MSMAC diminishes. We believe this is due to MSMAC's instruction-driven batch processing of MSM computations and the advantages offered by its powerful on-chip NoC. We recognize that our baseline lacks specific optimizations for the BN128 curve. For the MNT4753 curve, PipeZK is equipped with only 1 PADD unit, as the data width is 753 bits, which is quite large. cuZK is an MSM accelerator based on the Nvidia V100. Our baseline achieves speedup of up to 7.59 × and 12.01 × relative to these two works. Despite the variations in the number of PADD cores and operating frequencies among the compared works, after normalization, our baseline still leads across all comparisons in Tab. III. This further highlights the performance of the baseline we proposed.

TABLE III: Comparison of Our Proposed Baseline with Other Works on Different Elliptic Curves.

Curve		BN128[a]			MNT4753		
Work		MSMAC [14]	PipeZK [12]	**Baseline**	PipeZK [12]	cuZK [7]	**Baseline**
Window Size		13	4	13	4	16	13
Platform		Xilinx VP1502	UMC 28nm	TSMC 12nm	UMC 28nm	Nvidia V00	TSMC 12nm
Frequency		250 MHz	300 MHz	1 GHz	300 MHz	-	1 GHz
Time (ms)	2^{18}	10.65 (1.20 ×)	16 (1.80 ×)	8.9	184 (3.96 ×)	-	46.5
	2^{19}	19.45 (1.37 ×)	32 (2.25 ×)	14.2	369 (5.83 ×)	732 (11.7 ×)	63.3
	2^{20}	34.15 (**1.38** ×)	61 (**2.47** ×)	24.7	735 (**7.59** ×)	1163 (**12.01** ×)	96.8
	2^{21}	58.87 (1.29 ×)	-	45.6	-	1960 (11.91 ×)	164.5
	2^{22}	114.48 (1/31 ×)	-	87.4	-	3608 (11.97 ×)	301.3
	2^{23}	200.65 (1.17 ×)	-	171.5	-	6635 (11.47 ×)	578.7
	2^{24}	373.77 (1.10 ×)	-	339.3	-	-	1141.3
	2^{25}	736.68 (1.09 ×)	-	674.9	-	-	2282.8
	2^{26}	1420 (1.05 ×)	-	1346	-	-	4598.9

a. Both MSMAC and PipeZK utilize four PADD units.

B. Evaluation of Myosotis

To assess the comprehensive performance of Myosotis, including area, bandwidth, and latency, we construct four Myosotis cases based on the parameter configurations listed in Tab. I. The window size of the proposed baseline is 13, which is an optimal choice for balancing bucket area overhead and computational load. With this constraint, for the four cases of Myosotis, we set their window sizes to 12 and 11, respectively, aiming to keep the Bucket Array area of all compared cases as equal as possible or with no significant differences. We analyze the latency, area overhead, and bandwidth in detail in Fig. 12. Additionally, the Area-Latency Product (ALP), as shown in Eq. 6, is introduced to evaluate the cases of Myosotis, where a smaller ALP corresponds to better area efficiency or computation density. Average bandwidth can reflect whether the circuit is bandwidth-constrained.

$$\text{ALP} = \text{Area} \times \text{Latency}. \tag{6}$$

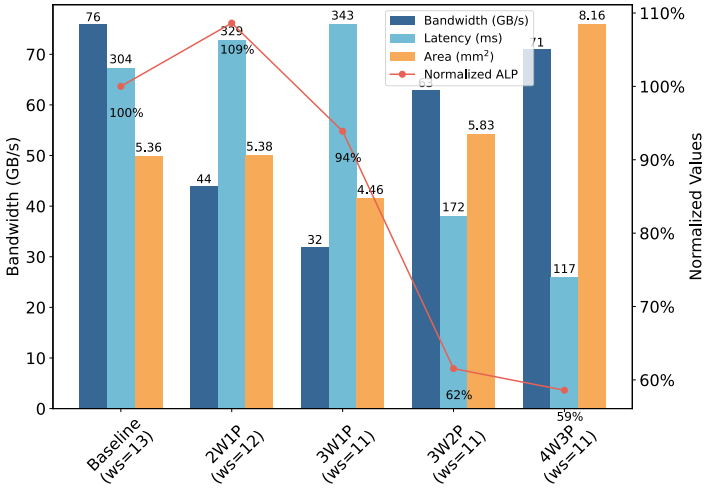

Fig. 12: Comparison of bandwidth, latency, area, and normalized ALP for different scenarios with similar Bucket capacities, where latency, area, and ALP are normalized to show relative values. 'X'W'Y'P represents 'X' window MSMs processed in parallel and 'Y' PADD units. The 'ws' represents the window size for each case. Bandwidth and latency are averaged over MSM degrees ranging from 2^{18} to 2^{26}.

For convenience, we use 'X'W'Y'P to represent a case with 'X' window MSMs processed in parallel and 'Y' PADD units in Myosotis. As a parameterized architecture, Myosotis follows the principle that 'X' is greater than 'Y'. So the primary contribution lies in reducing the architecture's bandwidth requirements. For instance, the performance and area of the 2W1P model are nearly identical to those of the baseline. However, the normalized ALP increases by 9%, while saving 43% of the bandwidth requirement. The performance of the 3W1P case is diminished by 13%, yet the reduction in overall area is made possible by a 25% reduction in bucket storage overhead. The normalized ALP is reduced by 6%, a relatively minor change, yet the bandwidth is significantly reduced by 58%. It is important to note that the experimental results may not align perfectly with our ideal outcomes. This discrepancy arises because the scalar length serves as a constraint and is not always an integer multiple of the window size, resulting in wasted computational resources during the final turn of the Bucket Classification step. We mitigate this inadequacy by selecting a suitable window size and ensuring backward compatibility within Myosotis.

Myosotis not only facilitates the parallel processing of multiple window MSMs but also enables the utilization of multiple PADD units to augment the computational capacity, with these supplementary PADD units able to share the majority of the original logic resources. The inclusion of integrated multi-PADD design and conflict-free arbitration logic substantially enhances the circuit's computational capabilities. The combination of these two factors results in higher area efficiency or computational density for the Myosotis. Compared to the 3W1P case, the 3W2P case shows a 50% decrease in latency and an overall 34% reduction in ALP.

In terms of bandwidth, when the number of window MSMs exceeds the number of PADD units, there is an improvement in average transmission bandwidth requirements. The increase in PADD units makes this improvement worse because lower latency means higher throughput. Taking the case 4W3P as an example, it has a significant 41% reduction in ALP, but only a 7% drop in transmission bandwidth requirements. We also do not recommend the parameter choice in Myosotis where the number of window MSMs equals the number of PADD units, because this choice is essentially consistent with the baseline we propose.

In contrast to previous work, which devoted a significant amount of resources to continually increasing Single Bucket Bank capacity to achieve marginal performance improvements, Myosotis represents a more optimal choice. This is because it strikes a balance between the increase in PADD units and the increase in caches, resulting in a notable improvement in area efficiency while also reducing bandwidth. We believe that simply increasing Bucket capacity shifts the workload to the Bucket Aggregation step and Window Aggregation step, as they need to handle more intermediate results from the Bucket Classification step.

Comparison when the overall area is approximately equal: Previously, we compare the metrics based on nearly equal bucket overhead. We are also curious about the performance differences in Myosotis cases with nearly identical overall areas but different configurations, particularly regarding the ALP metric, which indicates area efficiency. Furthermore, we compare three of these cases, where the total area of the MSM accelerator serves as a rough constraint, as shown in Tab. IV. The window sizes for the three cases vary, which is understandable because we aim to ensure their areas are roughly similar. Compared with our proposed baseline, in the 2W1P case, the ALP slightly increases, but the demand for interface bandwidth decreases by approximately 43%. The 3W2P case demonstrates a 38% improvement in area efficiency, coupled with a 17% decrease in bandwidth. This demonstrates that our proposed Myosotis can indeed explore and generate more computationally efficient MSM accelerators, which holds significant importance for the practical application of MSM accelerators.

TABLE IV: Comparison of the baseline and 2 cases in Myosotis with approximately equal area in Myosotis.

Case[1]	Baseline	2W1P	3W2P
Window Size	13	12	11
Bandwidth[b] (GB/s)	76.4 (1.00 ×)	**43.6** (**0.57** ×)	63.5 (0.83 ×)
Area (mm^2)	**5.36** (**1.00** ×)	5.38 (≈ 1.00 ×)	5.83 (1.09 ×)
Latency[b] (ms)	304 (1.00×)	329 (1.08 ×)	**172** (**0.57** ×)
Normalized ALP	100%	109%	**62%**

[a] 'X'W'Y'P represents 'X' window MSMs and 'Y' PADD units.
[b] Bandwidth and latency are averaged over MSM degrees ranging from 2^{18} to 2^{26}.

Observation of the relationship between throughput and operation intensity for different parameters: Furthermore, Myosotis is a parameterized design. We want to analyze the

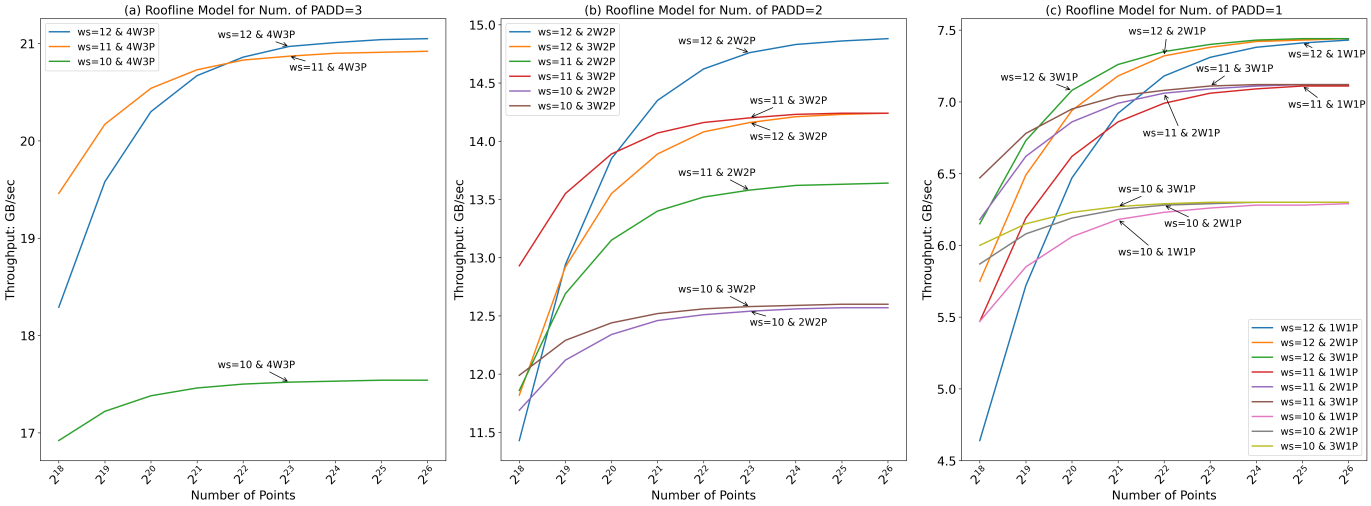

Fig. 13: Roofline Model of The Number of MSM Operation Points (i.e., Operational Intensity) and Myosotis's Throughput. We evaluate parameters including the number of windows, the number of PADD units, and window size. It is evident that the number of PADD units and window size have a significant impact on throughput. Throughput tends to stabilize as the degree of MSM increases, while the number of windows has a relatively minor effect on throughput. The number of windows primarily affects transmission bandwidth.

impact of each parameter on performance. Using a roofline model, we establish the relationship between throughput and operation intensity across different configurations, shown in Fig. 13, where operation intensity is measured by the number of points. We focused on three primary parameters: window size, the number of window MSMs, and the number of PADD units.

The roofline model shows that Myosotis's throughput rises with increasing operation intensity before stabilizing, as higher intensity maximizes pipeline utilization while minimizing the time ratios for bucket aggregation and window aggregation steps, reducing their impact on throughput. Window size and PADD unit quantity most directly affect throughput among the parameters. In Fig. 13, we divide the roofline model into three subfigures—(a), (b), and (c)—each corresponding to the number of PADD units of 3, 2, and 1, respectively. More PADD units lead to increased throughput, which aligns with our expectations. Within each subfigure, throughput generally rises with window size, as larger window sizes yield higher aggregation efficiency and fewer bucket classification turns, leading reduction in computational latency. This also implies a larger on-chip storage area, and the increase in area is exponential. However, the performance gain from this increase may be marginal. Therefore, when making comparisons, we also balance the number of window MSMs and the window size to align with practical feasibility. An exception is seen in Fig. 13 (b), where the case with ws=11&2W2P has 12 computation turns, while the case with ws=12&2W2P has only 11 turns, causing a noticeable throughput drop in the former. We attribute this to the scalar width limit of 253. Lastly, the number of window MSMs minimally impacts throughput, confirming earlier analyses that it primarily affects PADD unit utilization and Myosotis's transmission bandwidth.

VI. Conclusion

This paper introduces Myosotis, an efficiently pipelined parameterized architecture for Multi-Scalar Multiplication. Following principles of parameterization, Myosotis enhances performance by leveraging data sharing, optimizing cache usage, and integrating multi-PADD design compared to previous designs. We assess the area, latency, and their combination—ALP, indicating area efficiency, as well as the bandwidth of our proposed baseline and Myosotis with typical parameter selections. We suggest adjusting the parameters of Myosotis according to the specific requirements of the application, achieving a balance among these metrics to obtain the desired accelerator.

References

[1] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *Providing sound foundations for cryptography: On the work of shafi goldwasser and silvio micali*, 2019, pp. 203–225.

[2] R. Singh, A. D. Dwivedi, G. Srivastava, P. Chatterjee, and J. C.-W. Lin, "A privacy-preserving internet of things smart healthcare financial system," *IEEE Internet of Things Journal*, vol. 10, no. 21, pp. 18 452–18 460, 2023.

[3] H. Sun, T. Bai, J. Li, and H. Zhang, "zkdl: Efficient zero-knowledge proofs of deep learning training," Cryptology ePrint Archive, Paper 2023/1174, 2023. [Online]. Available: https://eprint.iacr.org/2023/1174

[4] B.-J. Chen, S. Waiwitlikhit, I. Stoica, and D. Kang, "Zkml: An optimizing system for ml inference in zero-knowledge proofs," 04 2024, pp. 560–574.

[5] H. Jin and J. Xiao, "Towards trustworthy blockchain systems in the era of "internet of value": development, challenges, and future trends," *Science China Information Sciences*, vol. 65, pp. 1–11, 2022.

[6] 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.

[7] 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.

[8] N. Pippenger, "On the evaluation of powers and related problems," in *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*. IEEE Computer Society, 1976, pp. 258–263.
[9] Wikipedia contributors, "Elliptic curve point multiplication," 2024, [Online; accessed 1-May-2024]. [Online]. Available: https://en.wikipedia.org/wiki/Elliptic_curve_point_multiplication
[10] C. Wang and M. Gao, "Sam: A scalable accelerator for number theoretic transform using multi-dimensional decomposition," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023, pp. 1–9.
[11] K. Aasaraai, D. Beaver, E. Cesena, R. Maganti, N. Stalder, and J. Varela, "Fpga acceleration of multi-scalar multiplication: Cyclonemsm," *Cryptology ePrint Archive*, 2022.
[12] 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)*. IEEE, 2021, pp. 416–428.
[13] C. Liu, H. Zhou, L. Yang, J. Xu, P. Dai, and F. Yang, "Gypsophila: A scalable and bandwidth-optimized multi-scalar multiplication architecture," ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3658259
[14] P. Qiu, G. Wu, T. Chu, C. Wei, R. Luo, Y. Yan, and H. Zhang, "Msmac: Accelerating multi-scalar multiplication for zero-knowledge proof," in *2024 61st ACM/IEEE Design Automation Conference (DAC)*, 2024.
[15] H. Zhou, C. Liu, L. Yang, L. Shang, and F. Yang, "Rezk: A highly reconfigurable accelerator for zero-knowledge proof," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–14, 2024.
[16] 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.*, jul 2024, just Accepted. [Online]. Available: https://doi.org/10.1145/3678006
[17] Y. Xu and D. Qian, "Zodiacmsm: A heterogeneous, multi-node and scalable multi-scalar multiplication system for zero knowledge proof acceleration," in *2023 IEEE 36th International System-on-Chip Conference (SOCC)*, 2023, pp. 1–6.
[18] C. F. Xavier, "Pipemsm: Hardware acceleration for multi-scalar multiplication," *Cryptology ePrint Archive*, 2022.
[19] 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
[20] C. Lab, "SuperScalar," https://github.com/z-prize/2023-entries/tree/main/prize-1-fpga-gpu-proof/prize-1a-msm/fpga/SuperScalar, accessed:2024-7-3.
[21] 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.
[22] 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*, 2023, pp. 340–353.
[23] X. Zhu, H. He, Z. Yang, Y. Deng, L. Zhao, and R. Hou, "Elastic msm: A fast, elastic and modular preprocessing technique for multi-scalar multiplication algorithm on gpus," Cryptology ePrint Archive, Paper 2024/057, 2024. [Online]. Available: https://eprint.iacr.org/2024/057
[24] C. Costello and B. Smith, "Montgomery curves and their arithmetic: The case of large characteristic fields," Cryptology ePrint Archive, Paper 2017/212, 2017. [Online]. Available: https://eprint.iacr.org/2017/212
[25] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters, "Twisted edwards curves," in *Progress in Cryptology–AFRICACRYPT 2008: First International Conference on Cryptology in Africa, Casablanca, Morocco, June 11-14, 2008. Proceedings 1*. Springer, 2008, pp. 389–405.
[26] C. Costello and B. Smith, "Montgomery curves and their arithmetic: The case of large characteristic fields," *Journal of Cryptographic Engineering*, vol. 8, no. 3, pp. 227–240, 2018.
[27] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *Journal of cryptographic engineering*, vol. 2, no. 2, pp. 77–89, 2012.
[28] W. Haddaji, L. Ghammam, N. E. Mrabet, and L. B. Abdelghani, "On computing the multidimensional scalar multiplication on elliptic curves," Cryptology ePrint Archive, Paper 2024/038, 2024, https://eprint.iacr.org/2024/038. [Online]. Available: https://eprint.iacr.org/2024/038
[29] 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
[30] P. Barrett, "Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor," in *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, ser. Lecture Notes in Computer Science, vol. 263. Springer, 1986, pp. 311–323.
[31] Y. Software, "Yrrid," https://github.com/z-prize/2023-entries/tree/main/prize-1-fpga-gpu-proof/prize-1a-msm/fpga/Yrrid, accessed:2024-7-7.
[32] Zprize, "ZPRIZE: ACCELERATING THE FUTURE OF ZERO KNOWLEDGE CRYPTOGRAPHY," https://www.zprize.io/, accessed:2024-7-8.

Changxu Liu received the B.S. degree in Microelectronics Science and Engineering from Wuhan University, Wuhan, China, in 2022. He is currently pursuing the Ph.D. degree with the State Key Laboratory of Integrated Chips and Systems, School of Microelectronics, Fudan University, Shanghai, China. His current research interests include hardware-software Co-design for privacy-preserving computing applications and digital IC design.

Hao Zhou received the B.S. degree in Electronic Science and Technology from Jilin University, Jilin, China, in 2017 and the M.S. degree in Electronics and Communication Engineering from University of Chinese Academy of Sciences, Beijing, China, in 2020. He is currently working toward the Ph.D. degree in Electronic and Information Engineering with the School of Microelectronics, Fudan University, Shanghai, China. His research interests include zero-knowledge proof, fully homomorphism encryption and VLSI implementation of digital systems.

Lan Yang received the B.S. degree in Microelectronic Science and Engineering from Fudan University, Shanghai, China, in 2023. She is currently pursuing the Ph.D. degree with the State Key Laboratory of Integrated Chips and Systems, School of Microelectronics, Fudan University, Shanghai, China. Her research interests include homomorphic encryption in privacy-preserving technologies and hardware acceleration.

Zheng Wu received the B.E. degree in Integrated Circuit Design and Integrated System from Huazhong University of Science and Technology, Wuhan, Hubei, China, in 2020. He is currently pursuing the Ph.D. degree with State Key Lab of Integrated Chips & System, Microelectronics Department, Fudan University, Shanghai, China. His current research interests include processor architecture design space exploration and automated deep learning compilation.

Patrick Dai is the founder of Semisand Chip Design Pte Ltd Singapore. Semichip team focus on ZKP algorithm research and engineering, and have already launched a variety of algorithm optimizing and accelerating solutions for ASIC, FPGA, GPU etc.

Yinlong Li is the senior FPGA engineer in hardware R&D center of Shanghai Academy of Future Internet Technology. His main research interests include FPGA-based hardware acceleration of cryptographic algorithms, blockchain and privacy computing, and zeroknowledge proof.

Shiyong Wu is the chief researcher in hardware R&D center of Shanghai Academy of Future Internet Technology. His main research interests include hardware acceleration of cryptographic algorithms, blockchain and privacy computing, hardware security, heterogeneous computing.

Fan Yang (Member, IEEE) received the B.S. degree from Xi'an Jiaotong University, Xi'an, China, in 2003, and the Ph.D. degree from Fudan University, Shanghai, China, in 2008. He is currently a Full Professor with the Microelectronics Department, Fudan University. His research interests include model order reduction, circuit simulation, high-level synthesis, acceleration of artificial neural networks, acceleration of privacy-preserving computing, and yield analysis and design for manufacturability.