Gypsophila: A Scalable and Bandwidth-Optimized Multi-Scalar Multiplication Architecture

Changxu Liu[1,2], Hao Zhou[1,2], Lan Yang[1,2], Jiamin Xu[1,2], Patrick Dai[3], Fan Yang[1,2]
[1] State Key Laboratory of Integrated Chips and Systems, Fudan University, China
[2] School of Microelectronics, Fudan University, China [3] Semisand Chip Design Pte. Ltd., Singapore

ABSTRACT

Multi-Scalar Multiplication (MSM) is a fundamental cryptographic primitive, which plays a crucial role in Zero-knowledge proof systems. In this paper, we optimize the single MSM Process Element (PE) utilizing buckets with fewer conflicts, enhanced by Greedy-based scheduling, to achieve higher efficiency. The evaluation results show our optimized single MSM PE achieving a speedup of over two times on average, peaking at 3.63 times compared to previous works. Furthermore, we introduce Gypsophila, a scalable and bandwidth-optimized architecture for implementing multiple MSM PEs. Leveraging the characteristics of the bucket method, we optimize the data flow by balancing the throughput of bucket classification, bucket aggregation, and result aggregation in MSM. Simultaneously, multiple PEs with different data access patterns share a universal point input channel and post-processing unit, which improves the module utilization and mitigates the bandwidth pressure. Gypsophila with 16 PEs, accomplishes 16 MSM tasks in a mere 1.01% additional time, showcasing an approximate 7.8% reduction in area, with only about $\frac{1}{16}$ of the bandwidth requirement, compared with 16 PEs without input channel and post-process unit sharing.

KEYWORDS

Multi-Scalar Multiplication, Zero-knowledge proofs, Hardware Acceleration, Parallel Computing

1 INTRODUCTION

Multi-Scalar Multiplication (MSM) represents an important task in cryptography, particularly in Zero-knowledge proofs (ZKP) [9]. The MSM operation with N degrees aims to multiply the scalars $\vec{a}$, a set of N integers $[a_0, a_1, \cdots, a_{N-1}]$, with a cyclic group $\mathbb{G}$ represented by a set of points $[G_0, G_1, \cdots, G_{N-1}]$ on an elliptic curve. It can be expressed as follows:

$$MSM(\vec{a}, \mathbb{G}) = a_0 \cdot G_0 + a_1 \cdot G_1 + \cdots + a_{N-1} \cdot G_{N-1}. \qquad (1)$$

While the algorithm may seem straightforward, point operations on elliptic curves are computationally concentrated. Moreover, with the increasing demand for applications in recent years, the degree of MSM has grown larger. The Pippenger algorithm [5, 15], also known as the bucket method, has been introduced into MSM. Decomposing the scalar enables the application of pipeline techniques to MSM, making it particularly effective in managing larger degrees of MSM.

In simple terms, the bucket method breaks down MSM into multiple reduced MSMs. Bucket classification and bucket aggregation steps are then performed on each reduced MSM to obtain their respective results. Finally, the results of the reduced MSMs are aggregated through a result aggregation step to obtain the final result. Further details are presented in Section 2.2.

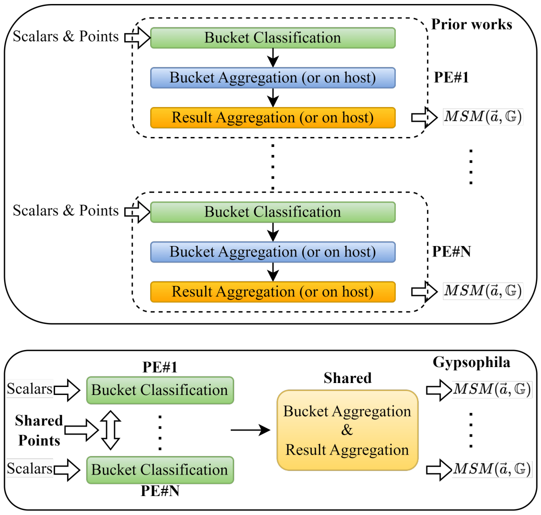
Scalars & Points
Bucket Classification
Prior works
Bucket Aggregation (or on host)
PE#1
Result Aggregation (or on host)
$MSM(\vec{a}, \mathbb{G})$
Scalars & Points
Bucket Classification
Bucket Aggregation (or on host)
PE#N
Result Aggregation (or on host)
$MSM(\vec{a}, \mathbb{G})$
PE#1
Gypsophila
Scalars
Bucket Classification
Shared
$MSM(\vec{a}, \mathbb{G})$
Shared Points
Bucket Aggregation & Result Aggregation
Scalars
Bucket Classification
$MSM(\vec{a}, \mathbb{G})$
PE#N

Figure 1: A Comparison of Gypsophila with Other Works.

Related Works: Numerous MSM acceleration schemes based on CPU, GPU, FPGA, and ASIC have been proposed. In [13], a bucket set construction is introduced, which can be applied to Pippenger's bucket method to accelerate MSM over fixed points through precomputation. The work [6] employs hand-written arm64 assembly for accelerating the finite field arithmetic in mobile applications. The work [12] and [14] present GPU implementations of MSM, which achieve favorable execution results. The work [17] was the first hardware implementation to leverage bucket methods for zk-SNARKs using ASIC. While it offered valuable insights, the design still leaned towards a conservative approach, which posed considerable bandwidth constraints. In [16], [1] and [2], FPGA implementations of MSM are presented. Despite numerous improvements, we believe there is room for enhancing their scheduling, suggesting additional opportunities to minimize pipeline bubbles and address data hazards. More importantly, these works primarily focus on a single MSM task. When confronted with multiple MSM tasks, their

Corresponding author: yangfan@fudan.edu.cn.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3658259

designs can only be straightforwardly replicated. The demands on bandwidth and resources increase linearly, and their architectures overlook scalability as well as dataflow optimization strategies for handling multiple MSM tasks. MSM dominates the proof generation stage in various concrete ZKP schemes [12, 13, 16]. With the increasing use of ZKP, there is an increasing demand for MSM computations. This underscores the need to optimize the architecture for multiple MSM PEs, serving as a key motivation for our work.

Our Contributions: In this paper, we introduce Gypsophila, an end-to-end scalable and bandwidth-optimized architecture for multiple MSM PEs. The proposed design is based on the bucket method, supporting parallel computation of multiple MSM PEs. Specifically, Gypsophila is meticulously optimized for scalability, aiming to alleviate bandwidth requirements and enhance data and resource utilization efficiency. Figure 1 provides a concise architectural overview, illustrating the distinctions between our work and other works when handling multiple MSM tasks. Other works typically use the same hardware for bucket classification, bucket aggregation, and result aggregation, or they may offload the latter two to the host. In contrast, our design divides these three steps into distinct modules. Balancing their execution times forms a pipeline between modules, improving overall throughput. Below are the details of our contributions:

- We optimize the MSM PE. Each pipelined MSM PE is a thread within our design, which is powered by a pipelined point adder (pPADD). We employ a combination of Queues and Buckets with fewer conflicts, coupled with Greedy-based scheduling, to minimize the bubbles in the pipeline and data hazards for the bucket classification in MSM.
- We propose Gypsophila, an architecture tailored for scenarios with multiple MSM PEs. Utilizing the characteristics of the bucket method, Gypsophila employs a streamlined dataflow organization by balancing the throughput of the bucket classification, bucket aggregation, and result aggregation within MSM. This allows Gypsophila to achieve PE-level scalability without compromising performance. Simultaneously, multiple PEs exhibit diverse data access patterns yet can share a universal point channel and post-processing unit, an area-efficient point adder (aPADD), optimizing module utilization and mitigating bandwidth pressure.
- We implement Gypsophila in Verilog HDL, and evaluate four versions, PE-1 (single PE), PE-2 (dual PEs), PE-4 (4 PEs) and PE-16 (16 PEs). The evaluation results demonstrate a significant improvement in the speed of a single MSM PE, surpassing previous work by 3.63x in time consumption and 5.06x in clock cycles. With 16 MSM PEs, the speed loss is minimal, reaching only 1.01% compared to the unoptimized linear expansion of MSM PEs. Through resource sharing, we achieve a substantial area reduction of approximately 7.8%.

2 BACKGROUND

2.1 Brief Introduction to Elliptic Curves

In mathematics, an elliptic curve E over F_q is the set of points that satisfy Equation 2.

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

The coefficients a_{sw} and b_{sw} must satisfy a certain condition $4a_{sw}^3 + 27b_{sw}^2 \neq 0$. Here, the parameters a_{sw} and b_{sw} serve exclusively as the elliptic curve's parameters. Equation 2 is commonly referred to as the Short Weierstrass curve. There are other curve forms, such as Montgomery curves and Twisted Edwards curves. In specific contexts, different curves are birationally equivalent [4, 7]. The Twisted Edwards curve, as shown in Equation 3, is preferred for its expedited computation, attributed to the efficiency of its addition formula. This efficiency arises from the curve's robust completeness property, enhancing its performance in cryptographic protocols.

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

The representation of points on an elliptic curve includes affine coordinates (x, y), as in Equations 2 and 3, and projective coordinates (X, Y, Z). Affine coordinates (x, y) are equivalent to $(X/Z, Y/Z)$. Extended coordinates under Twisted Edwards curves are discussed in [8, 10]. By introducing an auxiliary coordinate $T = XY$, (X, Y, Z, T) in extended projective coordinates can represent (X, Y, Z) in projective coordinates. The addition formula utilizing extended projective coordinates in the Twisted Edwards curves avoids inefficient modular inversion operations, achieving point addition with fewer modular multiplication operations. Furthermore, this addition formula is unified. In Equation 1, the addition of points G_i on the elliptic curve produces another point on the same elliptic curve.

2.2 Pippenger Algorithm

MSM is shown in Equation 1. Each $a_i \cdot G_i$ could be computed with:

$$\underbrace{G_i + G_i + \cdots + G_i}_{a_i \ times}. \tag{4}$$

The naive method to perform $a_i \cdot G_i$ is to use the double-and-add method with point addition and point double.

In Algorithm 1, we present a succinct overview of the bucket method tailored for MSM with N degree employing b-bit scalars.

Algorithm 1 Pippenger Algorithm

```
Init: set $S$, $R$, $MSM(\vec{a}, \mathbb{G})$ = NULL (Point at infinity)
BC: Bucket Classification
for $j = 0, j < m, j++$ do
    for $i = 0, i < N, i++$ do
        set $k = a_{ij}$                    ▷ Identify the target bucket.
        $S_{j,k} \leftarrow S_{j,k} + G_i$
    end for
end for
BA: Bucket Aggregation
for $j = 0, j < m, j++$ do
    for $k = 2^c - 1, k > 0, k--$ do
        $R_j \leftarrow R_j + S_{j,k}$
        $S_{j,k-1} \leftarrow S_{j,k} + S_{j,k-1}$
    end for
end for
RA: Result Aggregation
for $j = m - 1, j \geq 0, j--$ do
    $MSM(\vec{a}, \mathbb{G}) \leftarrow 2^c \cdot MSM(\vec{a}, \mathbb{G}) + R_j$
end for
```

By dividing MSM with b-bit scalars into $m = \lceil \frac{b}{c} \rceil$ groups of smaller reduced MSM with c-bit scalars, each a_i can be presented as:

$$(a_{i(m-1)}, \cdots, a_{ij}, \cdots, a_{i1}, a_{i0}). \tag{5}$$

A reduced scalar set is comprised of N elements, denoted as a_{ij}, where the index j remains constant. In conjunction with point groups, they constitute a reduced MSM.

$$S_k = \sum_{i=1}^{N} (a_{ij} == k) \cdot G_i. \tag{6}$$

$$R_j = \sum_{i=1}^{N} a_{ij} \cdot G_i = \sum_{k=0}^{2^c-1} k \cdot S_k. \tag{7}$$

$$MSM(\vec{a}, \mathbb{G}) = \sum_{i=1}^{N} a_i \cdot G_i = \sum_{j=0}^{m-1} \sum_{i=1}^{N} 2^{jc} \cdot a_{ij} \cdot G_i = \sum_{j=0}^{m-1} 2^{jc} \cdot R_j. \tag{8}$$

Bucket classification, as Equation 6 shows, classifies points into different buckets based on their indexes, involving $m(N-2^c)$ point addition operations. Bucket aggregation, as Equation 7 shows, aggregates these buckets to derive the results of reduced MSMs, requiring $m(2^{c+1})$ point addition operations. Finally, result aggregation, as Equation 8 shows, consolidates these results to obtain $MSM(\vec{a}, \mathbb{G})$, needing $b + m - c - 1$ point addition operations. Result aggregation can be decomposed, and implemented after the bucket aggregation of each reduced MSM.

3 PROPOSED DESIGN

In this section, we initially introduce the overall architecture of Gypsophila. We propose optimization strategies primarily focused on alleviating bandwidth and PPA metrics, given the dominance of bucket classification in MSM computation time. Then, we introduce MSM PE, responsible for performing bucket classification. Finally, we showcase the post-processing module, managing output from the MSM PE and performing both bucket aggregation and result aggregation.

3.1 Gypsophila

The comprehensive architecture of Gypsophila is illustrated in Figure 2. We employ a thread to represent an individual MSM PE. The number of threads is parameterized, and each thread is dedicated to a bucket classification. Each thread gets the scalars from the Scalar ping-pong Buffer, and the points from Shared Point Buffer (S-P Buffer). For MSM, the scalars are variable while the points are relatively constant. To maximize the reuse of these points and alleviate the bandwidth pressure of MSM (considering the substantial overhead of points), multiple threads can share the input points and operate in a highly synchronized manner. The input data only needs to increase by the bit width of an index as the number of PEs grows. This approach significantly reduces the previously substantial increase in bandwidth. However, the S-P Buffer cannot be independently read by multiple threads according to their own data access patterns. Our solution involves incorporating register slices and FIFOs on the data path from the S-P Buffer to each thread, acting as a "virtual" individual point buffer for each thread. This arrangement allows each thread to operate as if it has an independent point buffer, reducing data dependencies between threads in this architecture. Based on our assessment, utilizing the "virtual" point buffer design significantly increases point reuse. However, the synchronization of points between different channels also introduces additional time overhead, which is analyzed in Section 4.2. Thus, concurrent processing of all threads is feasible, and this also contributes to improving timing in the actual layout and routing. We also configure the Scalar Buffer in a ping-pong buffer structure to mask the latency of data input.

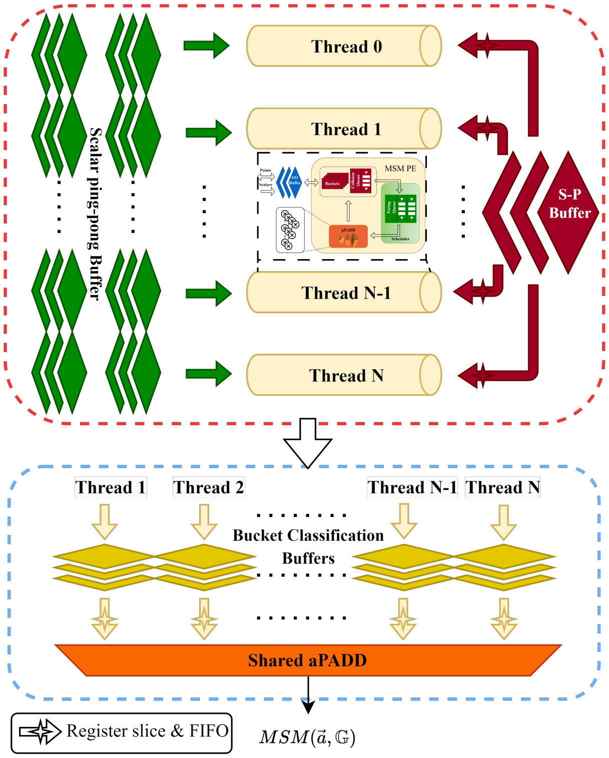

Figure 2: Overall architecture of Gypsophila. The red dashed box above represents the bucket classification, while the blue dashed box below signifies the bucket aggregation and result aggregation.

The design of Gypsophila is organized with streamlined dataflow, as illustrated in Figure 3. We employ Bucket Classification (BC) Buffers in Figure 2 to capture thread outputs and relay them to the shared aPADD. In that case, threads can proceed to the next round of calculations as soon as possible. Both [17] and [2] delegate this task to the host. Work [1] does not provide a clear solution. While work [16] employs the same EC Adder to perform this computation, it results in redundant performance for pPADD and compromises data throughput.

Multiple threads initially store their outcomes in distinct BC Buffers. These threads collaboratively access a Shared aPADD to execute the bucket aggregation and result aggregation. More details about aPADD will be introduced in Section 3.3. Given that the latency of the bucket classification dominates the computation time in MSM, the Shared aPADD effectively capitalizes on this period to manage the bucket aggregation and result aggregation for numerous threads. The arbitration logic is determined by whichever thread finishes first. To maintain a balanced execution time, it is feasible to utilize more Shared aPADDs in specific contexts. While this introduces additional overhead, the added costs are minimal. The data path from the BC Buffer to the Shared aPADD incorporates both register slices and FIFOs, contributing to enhancements in layout and routing efficiency. Increasing the number of PEs can simultaneously increase data throughput, highlighting the performance enhancement brought about by Gypsophila's scalability.

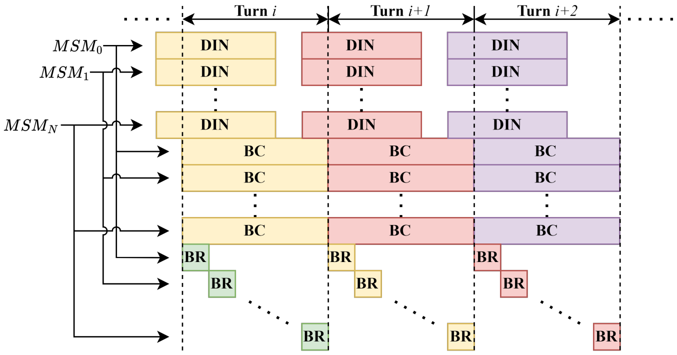

Figure 3: The dataflow organization of Gypsophila. The same color indicates membership to the same reduced MSM. DIN represents data input, BC is bucket classification, and BR stands for both bucket aggregation and result aggregation.

3.2 MSM PE

The MSM PE serves as the fundamental thread in our architecture and is responsible for executing the bucket classification within MSM. In the bucket classification of the Pippenger Algorithm, both points and scalars, depicted as a set of indexes based on radix 2^c, are fetched from the external DRAM and relayed through the I/O buffer. The results, a group of points will be sent back through the I/O buffer.

Figure. 4 shows the overview microarchitecture of MSM PE. Each time, points are transferred into Buckets (consisting of multiple slots) addressed by the corresponding indexes, and the corresponding point slots are marked to *busy*. When another point with the same index tries to take up the point slot marked as *busy*, the scheduler matches these two points as a pairing of points and sends this pair into the Pairing Queue. pPADD is fed by Pairing Queue, and it could handle a complete point addition operation with a strong intensive pipeline. The result of pPADD will be stored back to the corresponding point slot. However, due to the random distribution of scalars, inevitable conflicts will arise when the result of pPADD and point from I/O Buffer attempt to simultaneously access to, or pair within the Buckets, which are implemented with SRAMs. We use the Collision Queue to store these conflicting points for processing during idle intervals.

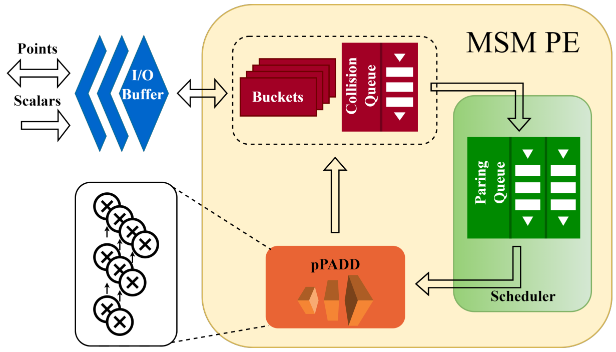

Figure 4: Microarchitecture overview of a single MSM PE.

We propose two methods to mitigate conflicts during this process while also minimizing pipeline bubbles in pPADD. First, the scheduler advocates for integrating Greedy-based scheduling, allowing the immediate pairing of points when their indexes align. This is facilitated by a fully pipelined point adder (pPADD), capable of supporting point addition in extended coordinates and complemented by a spacious Pairing Queue. Secondly, we introduce Buckets with fewer conflicts. Conflicts are confined to each bank of Buckets, significantly reducing the likelihood of entries into the Collision Queue and thereby minimizing pipeline bubbles of pPADD. Buckets with fewer conflicts enhance the performance of the bucket classification.

We construct the fully pipelined PADD with 9 modular multiplication units [10]. We assert that the use of MixedAdd [1, 8] could lead to data dependencies in the bucket classification. To mitigate this, we recommend adopting a point addition algorithm in extended coordinates, as this will help to break data dependencies and minimize pipeline bubbles. Furthermore, we employ a 4-layer Karatsuba method [11] and the Barrett reduction algorithm [3, 16] to optimize the area of the modular multiplication unit. The ωNAF technique [1] is also used to reduce the area overhead of Buckets.

3.3 Bucket Aggregation & Result Aggregation

Post-processing, comprising both bucket aggregation and result aggregation, must be conducted for a set of points within the buckets of the MSM PE after bucket classification. In bucket classification, pPADD efficiently processes incoming points in a pipelined manner for high performance. However, in bucket aggregation and result aggregation, performance is hindered by dependencies among post-processed points, preventing pipelined processing. This leads to performance inefficiencies and potential backpressure on the input of the next data round, ultimately reducing throughput.

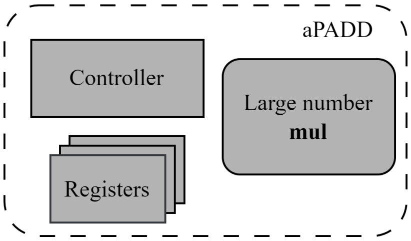

Figure 5: Simple structure of aPADD.

We introduce an area-efficient PADD (aPADD) to handle both bucket aggregation and result aggregation, preventing it from becoming a bottleneck that hampers the overall throughput of the

Table 1: Resource and Power Consumption for Gypsophila

Design	Platform	Area/Resource	Power
PE-1(377w)	Xilinx Virtex UltraScale+ XCVU13P (200MHz)	(1469k(CLBs)/ 3684(DSPs)	48.91W
PE-2(377w)		2783k(CLBs)/ 7122(DSPs)	95.12W
PE-4(256w)		2462k(CLBs)/ 6858(DSPs)	81.04W
PE-16(377w)	TSMC 12nm (1GHz[1])	79.80 mm^2	45.25W

[1] We set the target frequency to 1GHz. The results indicate that the critical path is less than 1ns, demonstrating significant potential for achieving higher frequency.

design while minimizing cost. This is crucial as Gypsophila adopts a streamlined dataflow organization. Figure 5 showcases the aPADD design. By leveraging multiple registers and a Large number multiplication unit (**mul**), combined with the controlling logic, we have constructed an aPADD that offers comparable latency to pPADD.

Without pipelining and based on the point addition formula, the core of aPADD is a **mul** that is half the size of the curve's word length. For a 377-bit width point, we only need a 189-bit **mul**, which is about $\frac{1}{3}$ the size of the modular multiplication unit used in pPADD using the Karatsuba method.

4 EVALUATION

4.1 Methodology

In this section, we present the implementation results of Gypsophila along with pertinent analysis. Gypsophila is implemented in Verilog HDL, and this architecture is utilized to generate four sets of distinct hardware for evaluation: PE-1 (single PE), PE-2 (dual PEs), PE-4 (4 PEs) and PE-16 (16 PEs). We set the bit-width of the index to 16 for all versions except PE-16, where the index is 12-bit.

Targeting the MSM on the BLS12-377 curve, we synthesize PE-1 and PE-2 versions on FPGA using the Xilinx Vivado 2022.1 tools. As part of our evaluation, we synthesize the PE-4 version on FPGA targeting curves such as BN256, since there aren't sufficient resources on the FPGA to accommodate four 377-bit level MSM PEs. Nonetheless, this provides a 256-bit security level and practical applicability. The largest configuration, PE-16, designed for the BLS12-377 curve, has been synthesized utilizing the TSMC 12nm process. Resource/area reports and power consumption reports for each of these implementations are presented in Table 1. In this regard, the resource utilization for a single PE on FPGA (solely responsible for the bucket classification) amounts to approximately 692k LUTs, 620k REGs, 3438 DSPs, 288 URAMs, and 684 BRAMs. The largest version, PE-16, has an estimated area of approximately 79.80 mm^2 on the TSMC 12nm process, according to the synthesis report from Synopsys Design Compiler.

4.2 Performance Results

We provide a performance comparison of our work with other custom designs in Table 2. The range of MSM degrees encompasses typical application scenarios, spanning from 2^{18} to 2^{25}. It's worth noting that apart from PipeZK, which is a 28nm ASIC designed using the UMC process, all other works included in this comparison are evaluated on Xilinx FPGAs at the 16nm level. The average throughput of other works in the Table 2 are derived from our analysis of the data provided in their respective papers.

In MSM with degrees ranging from 2^{18} to 2^{25}, our design demonstrates an average speedup of over two times compared to existing works, whether measured in the number of clock cycles or time consumption. At 2^{21}, the speedup relative to existing works reaches 3.63 times in time consumption and 5.06 times in the number of clock cycles. With an average throughput of 1.71 GB/s, our single PE surpasses that of other implementations. Although our operating frequency is limited to 200 MHz, it is essential to highlight that Gypsophila is meticulously designed with ASIC as its primary target platform. We have not pursued dedicated optimizations for FPGA at this time. We believe that with further optimizations tailored for FPGAs, Gypsophila can achieve lower resource overhead and higher operating frequencies, leading to better PPA metrics.

We also conduct evaluations on PE-2, PE-4, and PE-16 versions. The results highlight Gypsophila's proficiency in parallelizing the processing of multiple MSM tasks. This bandwidth-optimized architecture incurs only marginal speed loss. Table 3 shows the speed loss and area reduction of three versions compared to the unoptimized simple replicated MSM (PE-1) design. For fairness, this comparison is based on the number of clock cycles. The table shows the speedup loss when performing multiple MSM tasks synchronously in a pipelined fashion, compared to the time required for a single MSM task. Since the primary computational load for MSM is concentrated in the bucket classification, the time for computing MSM in the pipelined fashion aligns with the time required for the bucket classification. Table 3 also shows that Gypsophila reduces the area through resource sharing. With 16 PEs, it reduces the area by about 7.8% compared to the unoptimized architecture. It is evident that Gypsophila's scalable architecture, while conserving bandwidth, enhancing throughput, and reusing points, has minimal impact on performance. Even when performing calculations for MSM with 2^{18} in PE-16, it results in a mere 1.01% maximum increase in time consumption. However, its bandwidth requirements have almost remained unchanged compared to a single MSM PE. In contrast to the unoptimized use of 16 individual MSMs, its bandwidth is only 1/16 of the latter, yet it retains nearly identical computational capability and throughput. We also observe that, in general, the lower the degree of MSM, the more noticeable the speed loss becomes. This is because more MSM stalls are caused by internal point conflicts, especially bubbles in the pPADD pipeline. As the degree increases, the absolute count of these bubbles also rises, and the impact of point synchronization between multiple PEs becomes relatively smaller. The speed loss from point synchronization is influenced by the scalar arrangement order, leading to potentially varying values in specific cases. However, overall, the speed loss is minimal and follows statistical patterns.

5 CONCLUSION

In this paper, we present Gypsophila, a scalable and bandwidth-optimized multi-scalar multiplication architecture. The MSM PE is designed with a novel bucket method and Greedy-based scheduling. Leveraging Buckets with fewer conflicts, the MSM PE achieves higher data throughput than other works. Gypsophila, featuring

Table 2: Performance Comparison of Single MSM PE with Other Works

Design	Frequency	# Clock Cycles/Time (ms)								Throughput (Avg)
		2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}	
PipeMSM[16]	125MHz	8.6M /68.8	17.1M /136.6	34.1M /273.0	-	-	-	-	-	421.6MB/s
CycloneMSM[1]	250MHz	-	-	-	-	204.5M /817.9	283.3M /1133	440.3 /1761	754M /3016	1.34GB/s
Hardcaml[2]	278MHz	-	138.7M /499	150.1M /540	172.4M /620	216.8M /780	304.1M /1094	-	-	578.0MB/s
PipeZK[17][1]	300MHz (28nm)	27.6M /92	55.2M /184	110.4M /368	-	-	-	-	-	316.8MB/s
Ours	200MHz	4.8M **(1.80x)** /23.8 **(2.89x)**	9.0M **(1.90x)** /44.8 **(3.05x)**	17.4M **(1.96x)** /86.8 **(3.15x)**	34.1M **(5.06x)** /170.7 **(3.63x)**	67.7M **(3.02x)** /338.6 **(2.30x)**	134.9M **(2.10x)** /674.3 **(1.62x)**	269.2M **(1.64x)** /1346 **(1.31x)**	537.8M **(1.4x)** /2689 **(1.12x)**	**1.71GB/s**

[1] The paper presents it as being based on the BLS12-381 protocol, and here, we consider that it exhibits only minor differences in data volume compared to BLS12-377, allowing for an approximate comparison.

Table 3: Gypsophila's Speedup Loss and Area Reduction.[1]

Degree	Speedup Loss[2]			Area Reduction[3]		
	PE-2	PE-4	PE-16	PE-2	PE-4	PE-16
2^{18}	0.23%	0.51%	1.01%	4.3%	6.3%	7.8%
2^{19}	0.11%	0.20%	0.50%			
2^{20}	0.05%	0.13%	0.32%			
2^{21}	0.04%	0.08%	0.21%			
2^{22}	0.02%	0.06%	0.16%			
2^{23}	≍0%	≍0%	0.07%			
2^{24}	≍0%	0.04%	0.11%			
2^{25}	0.02%	0.04%	0.11%			

[1] Bit-width of the index is 12.
[2] "≍0%" implies less than 0.01%.
[3] The data is derived from four versions implemented in the TSMC 12nm process, with PE-1 occupying an area of about 5.41 mm^2.

multiple MSM PEs, implements a streamlined data flow by optimizing the throughput of bucket classification, bucket aggregation, and result aggregation within MSM. Resources sharing and data reuse effectively alleviate the bandwidth pressure on data transmission and enhance data throughput. Furthermore, Gypsophila provides scalability and can be parameterized to meet specific application requirements.

ACKNOWLEDGMENTS

This work was supported partly by National Key R&D Program of China 2023YFB2704600, partly by National Natural Science Foundation of China (NSFC) research projects 92373207 and 62090025.

REFERENCES

[1] Kaveh Aasaraai, Don Beaver, Emanuele Cesena, Rahul Maganti, Nicolas Stalder, and Javier Varela. 2022. Fpga acceleration of multi-scalar multiplication: Cyclonemsm. *Cryptology ePrint Archive* (2022).

[2] Ben Devlin Andy Ray. [n. d.]. HARDCAML. https://zprize.hardcaml.com/msm-overview.html. Accessed:2023-10-15.

[3] Paul Barrett. 1986. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 311–323.

[4] Daniel J Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. 2008. 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, 389–405.

[5] Daniel J Bernstein, Jeroen Doumen, Tanja Lange, and Jan-Jaap Oosterwijk. 2012. Faster batch forgery identification. In *Progress in Cryptology-INDOCRYPT 2012: 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings 13*. Springer, 454–473.

[6] Gautam Botrel and Youssef El Housni. [n. d.]. EdMSM: Multi-Scalar-Multiplication for recursive SNARKs and more; EdMSM: Multi-Scalar-Multiplication for recursive SNARKs and more. ([n. d.]). https://www.zprize.io/

[7] Craig Costello and Benjamin Smith. 2018. Montgomery curves and their arithmetic: The case of large characteristic fields. *Journal of Cryptographic Engineering* 8, 3 (2018), 227–240.

[8] Tanja Lange Daniel J. Bernstein. [n. d.]. Explicit-Formulas Database. https://www.hyperelliptic.org/EFD/.

[9] Shafi Goldwasser, Silvio Micali, and Chales Rackoff. 2019. The knowledge complexity of interactive proof-systems. In *Providing sound foundations for cryptography: On the work of shafi goldwasser and silvio micali*. 203–225.

[10] Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson. 2008. Twisted Edwards Curves Revisited. Cryptology ePrint Archive, Paper 2008/522. https://eprint.iacr.org/2008/522 https://eprint.iacr.org/2008/522.

[11] Anatolii Alekseevich Karatsuba and Yu P Ofman. 1962. Multiplication of many-digital numbers by automatic computers. In *Doklady Akademii Nauk*, Vol. 145. Russian Academy of Sciences, 293–294.

[12] Tao Lu, Chengkun Wei, Ruijing Yu, Chaochao Chen, Wenjing Fang, Lei Wang, Zeke Wang, and Wenzhi Chen. 2022. Cuzk: Accelerating zero-knowledge proof with a faster parallel multi-scalar multiplication algorithm on gpus. *Cryptology ePrint Archive* (2022).

[13] Guiwen Luo, Shihui Fu, and Guang Gong. 2023. Speeding up multi-scalar multiplication over fixed points towards efficient zksnarks. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023), 358–380.

[14] Weiliang Ma, Qian Xiong, Xuanhua Shi, Xiaosong Ma, Hai Jin, Haozhao Kuang, Mingyu Gao, Ye Zhang, Haichen Shen, and Weifang Hu. 2023. GZKP: A GPU Accelerated Zero-Knowledge Proof System. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 340–353.

[15] Nicholas Pippenger. 1976. On the evaluation of powers and related problems. In *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*. IEEE Computer Society, 258–263.

[16] Charles F Xavier. 2022. Pipemsm: Hardware acceleration for multi-scalar multiplication. *Cryptology ePrint Archive* (2022).

[17] Ye Zhang, Shuo Wang, Xian Zhang, Jiangbin Dong, Xingzhong Mao, Fan Long, Cong Wang, Dong Zhou, Mingyu Gao, and Guangyu Sun. 2021. Pipezk: Accelerating zero-knowledge proof with a pipelined architecture. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 416–428.