SCALES: SCALable and Area-Efficient Systolic Accelerator for Ternary Polynomial Multiplication

Samuel Coulon, Tianyou Bao, and Jiafeng Xie, *Senior Member, IEEE*

***Abstract*—Polynomial multiplication is a key component in many post-quantum cryptography and homomorphic encryption schemes. One recurring variation, ternary polynomial multiplication over ring $\mathbb{Z}_q/(x^n+1)$ where one input polynomial has ternary coefficients {−1,0,1} and the other has large integer coefficients {0, $q-1$}, has recently drawn significant attention from various communities. Following this trend, this paper presents a novel SCALable and area-Efficient Systolic (SCALES) accelerator for ternary polynomial multiplication. In total, we have carried out three layers of coherent interdependent efforts. First, we have rigorously derived a novel block-processing strategy and algorithm based on the schoolbook method for polynomial multiplication. Then, we have innovatively implemented the proposed algorithm as the SCALES accelerator with the help of a number of field-programmable gate array (FPGA)-oriented optimization techniques. Lastly, we have conducted a thorough implementation analysis to showcase the efficiency of the proposed accelerator. The comparison demonstrated that the SCALES accelerator has at least 19.0% and 23.8% less equivalent area-time product (eATP) than the state-of-the-art designs. We hope this work can stimulate continued research in the field.**

***Index Terms*—Area-efficient, block-processing, FPGA, scalable, systolic hardware accelerator, ternary polynomial multiplication.**

I. Introduction

Ternary polynomial multiplication has attracted substantial attention from the research community recently as it can be used in many critical cryptographic applications such as the Brakerski/Fan-Vercauteren (BFV) homomorphic encryption and decryption procedures [1]. Meanwhile, the research community is also exploring effective methods to implement this type of polynomial multiplication on different platforms for practical usage, particularly on hardware platforms like field-programmable gate arrays (FPGAs) [2].

Prior/Recent Works: Indeed, efficient FPGA acceleration of ternary polynomial multiplication has been an interesting and hot research topic recently [2], [3]. Due to its specific parameter setup, where one input polynomial has ternary coefficients {−1,0,1} and another one involves large integer coefficients (e.g., 32-bit, as seen in [1], [4]), there exist two types of implementation strategies, i.e., the number theoretic transform (NTT)-based method [3], [5] and the schoolbook (or its variants)-related approach [2], [6].

Challenge and Motivation: Each of the above-menti- oned methods has its unique advantages over the others. The NTT-based method features lower time complexity, while the schoolbook approach typically involves simpler structural setup. Although modern FPGAs offer resources for implementing computationally intensive operations, efficiently deploying these resources to implement the accelerator according to its parameter setting still requires further exploration. For example, the NTT-based works often employ DSPs for multiplication, while the schoolbook-based ones do not. Finally, as different applications may set different resource usage requirements, a structural- and processing-scalable (yet efficient) accelerator is also needed.

Proposal and Major Contributions: We noticed that systolic structure represents a useful design style for the targeted polynomial multiplication [2] because of its unique features like regularity [7], [8]. On the other hand, however, systolic design typically involves large area usage, and designing an area-efficient systolic accelerator is challenging. With this consideration, in this paper, we propose a novel **SCAL**able and area-**E**fficient **S**ystolic (SCALES) accelerator for ternary polynomial multiplication. Key contributions:

- We have derived a new block-processing algorithm for scalable and systolic polynomial multiplication.
- We have then innovatively designed the proposed algorithm into a novel SCALES accelerator.
- We have finally conducted a thorough evaluation to showcase its superior performance.

The rest of the paper is organized as follows. Preliminaries and the algorithm are presented in Section II. The SCALES accelerator is proposed in Section III. Evaluation and conclusion are given in Sections IV and V, respectively.

II. Preliminaries and Proposed Algorithm

Notations: Notations used are: (i) modulus $q = 2^k$ (k: coefficient bit-width); (ii) n is the polynomial degree; (iii) the scalable matrices are of size $v \times u$; (iv) B, D, and T represent polynomials used in ternary polynomial multiplication, with their respective coefficients b_i, d_i, and t_i.

Targeted Ternary Polynomial Multiplication: Without loss of generality, we use the ternary polynomial multiplication used in the original BFV scheme [1], [4] as the study case. In particular, one polynomial has coefficients in Ring $\mathcal{R}_2$, and another polynomial has coefficients in Ring $\mathcal{R}_q$, with the product polynomial's coefficients also in $\mathcal{R}_q$.

Consideration: As mentioned above, the two input polynomials involve unequal-sized coefficients. In this situation, we consider two strategies: (i) the widely used NTT strategy is

Received 11 August 2024; accepted 17 November 2024. Date of publication 25 November 2024; date of current version 16 December 2024. The work of Jiafeng Xie was supported in part by NIST under Grant 60NANB20D203 and in part by NSF under Grant SaTC-2020625. The work of Samuel Coulon was supported by the REU supplement of NSF under Grant SaTC-2020625. *(Corresponding author: Jiafeng Xie.)*

The authors are with the Department of Electrical and Computer Engineering, Villanova University, Villanova, PA 19085 USA (e-mail: scoulon@villanova.edu; tbao@villanova.edu; jiafeng.xie@villanova.edu).

Digital Object Identifier 10.1109/LCA.2024.3505872

applicable but requires a prime q to satisfy $q \equiv 1 \pmod{2n}$ (according to [9]), extra resources spent on the modular reduction, and coefficient-size extension of input B (from $\mathcal{R}_2$ to $\mathcal{R}_q$) [3], [5]; (ii) the schoolbook method (or similar) offers the advantage of a power of 2 modulus (free modular reduction) [2], [6] and cheap ternary multiplication.

Note that we do not consider traditional fast algorithms like Karatsuba [8], [10] as it requires the splitting of the polynomials into sub-polynomials for addition first, which may increase the size of the small coefficient involved operations (point-wise multiplications) and offset the original gain.

Proposed Strategy: We work from the schoolbook method and propose to: (a) take advantage of ternary coefficients (without extension from $\mathcal{R}_2$ to $\mathcal{R}_q$) to design efficient multiplication structures; (b) set modulus q as a power of 2 to utilize free modular reduction, the same as [2]; (c) develop a novel strategy to process multiplication in smaller sub-matrices (block-wise) instead of by element or by column.

Definition: We define the polynomial multiplication as:

$$T = BD \bmod x^n + 1, \tag{1}$$

where $B = \sum_{i=0}^{n-1} b_i x^i$, $D = \sum_{i=0}^{n-1} d_i x^i$, and $T = \sum_{i=0}^{n-1} t_i x^i$ (t_i and d_i are $\log_2 q$-bit integers and $b_i \in \{-1, 0, 1\}$).

We can then have $T = \sum_{i=0}^{n-1} d_i x^i B \bmod f(x) = \sum_{i=0}^{n-1} d_i B^{(i)}$, where $B^{(i)} = x^i B \bmod f(x)$ and $B^{(0)} = B$.

Define again $B^{(i)} = \sum_{j=0}^{n-1} b_j^{(i)} x^j$, we can have $B^{(i+1)} = x^{i+1} B \bmod f(x) = B^{(i)} x = b_0^{(i)} x + b_1^{(i)} x^2 + \cdots + b_{n-1}^{(i)} x^n$, which can be substituted with $x^n \equiv -1$ to have $B^{(i+1)} = -b_{n-1}^{(i)} + b_0^{(i)} x + b_1^{(i)} x^2 + \cdots + b_{n-2}^{(i)} x^{n-1}$.

Meanwhile, as $B^{(i+1)} = b_0^{(i+1)} + \cdots + b_{n-1}^{(i+1)} x^{n-1}$, which can be used to compare with the above $B^{(i+1)}$ to have $b_0^{(i+1)} = -b_{n-1}^{(i)}$, $b_1^{(i+1)} = b_0^{(i)}$, $\ldots$ $b_{n-1}^{(i+1)} = b_{n-2}^{(i)}$, which can be used to transform (1) into the form of

$$\begin{bmatrix} t_0 \\ t_1 \\ \vdots \\ t_{n-1} \end{bmatrix} = \begin{bmatrix} b_0 & -b_{n-1} & \cdots & -b_1 \\ b_1 & b_0 & \cdots & -b_2 \\ \vdots & \vdots & \ddots & \vdots \\ b_{n-1} & b_{n-2} & \cdots & b_0 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{n-1} \end{bmatrix}, \tag{2}$$

which can be rewritten as $[T]^{n \times 1} = [B]^{n \times n} \times [D]^{n \times 1}$.

One can transfer (2) into column-wise-based accumulation to obtain fast computation. However, as n is very large, e.g., $n = 4,096$, the accelerator will involve a large area, and meanwhile, it is hard for further scalable processing.

Further Strategy: We thus propose to compute (2) as: (a) decompose the main matrix $[B]^{n \times n}$ into a number of smaller-size matrices (size of $v \times u$); (b) transfer (2) into the accumulation of these sub-matrix-vector products for scalable operation (each sub-matrix is scalable); (c) execute the whole polynomial multiplication with systolic processing.

Therefore, we can rewrite (2) as (v and u are integers)

$$\begin{bmatrix} T_0 \\ T_1 \\ \vdots \\ T_{s-1} \end{bmatrix} = \begin{bmatrix} B_{0,0} & B_{0,1} & \cdots & B_{0,w-1} \\ B_{1,0} & B_{1,1} & \cdots & B_{1,w-1} \\ \vdots & \vdots & \ddots & \vdots \\ B_{s-1,0} & B_{s-1,1} & \cdots & B_{s-1,w-1} \end{bmatrix} \begin{bmatrix} D_0 \\ D_1 \\ \vdots \\ D_{w-1} \end{bmatrix}, \tag{3}$$

Algorithm 1: Proposed Algorithm for Polynomial Multiplication of BFV.

```
Input  : $B$ and $D$ ($D$ is an integer polynomial; $B$ is a
         ternary polynomial;
Output: $T = BD \bmod (x^n + 1) \bmod q$;
Initialization step
1  decompose the polynomials into respective
     sub-vectors/matrices (size of $v \times u$)
Main step
2  for $i = 0$ to $s - 1$ do
3      $\overline{T} = 0$; // $v$-size vector of $k$-bit values
4      for $j = 0$ to $w - 1$ do
5          $\overline{T} = \overline{T} + [B_{i,j}][D_j]$;
6      end
7      $T_i = \overline{T}$; // intermediate result $T_i$ delivered out
8  end
Final step
```

where $n = u \times w$ and $n = v \times s$; $[T_i]$ is a v-length vector ($0 \le i \le s-1$), e.g., $[T_0] = [t_0\, t_1 \ldots t_{v-1}]^T$; $[D_j]$ is a u-length vector ($0 \le j \le w-1$), e.g., $[D_0] = [d_0\, d_1 \ldots d_{u-1}]^T$; $[B_{i,j}]$ is a $v \times u$ matrix ($0 \le j \le w-1$), e.g.,

$$\left[B_{0,0}\right] = \begin{bmatrix} b_0 & -b_{n-1} & \cdots & -b_{n-u+1} \\ b_1 & b_0 & \cdots & -b_{n-u+2} \\ \vdots & \vdots & \ddots & \vdots \\ b_{v-1} & b_{v-2} & \cdots & \ddots \end{bmatrix}. \tag{4}$$

Following the proposed processing strategy, we can have

$$[T_0] = [B_{0,0}][D_0] + [B_{0,1}][D_1] + \cdots + [B_{0,w-1}][D_{w-1}], \tag{5}$$

which similarly applies to the calculation of $[T_1]$, …, $[T_{s-1}]$. From (5), one can see (2) is transferred into the calculation of s number of v-length vectors of $[T_0]$, …, $[T_{s-1}]$, where each $[T_i]$ can be obtained through the accumulation of w number of $v \times u$ matrix-vector products ($[B_{i,j}][D_j]$).

The above process is summarized in Algorithm 1. Overall, this algorithm fulfills the proposed strategy: (i) the elements contained in each $[B_{i,j}]$ are ternary coefficients, which involves small resource usage (see Section IV); (ii) the accumulation of these matrix-vector products ($[B_{i,j}][D_j]$) can be realized by systolic processing; (iii) the size of the sub-matrix is flexible for scalable implementation.

III. Proposed Accelerator: SCALES

Overall Design Strategy: The proposed SCALES accelerator is shown in Fig. 1, which contains: D-Data BRAMs (D-RAM), B-Data Shift Registers (B-REG), Processing Element Chain (PEC), Accumulator and Data Shift-Out Component (ACC), and Control Unit (CU). Their details and specific FPGA-oriented design techniques are given below.

D-RAM: This component is a u-size array of BRAMs (each BRAM is $width = k$, $depth = n/u$) responsible for storing coefficients of D (d_i) and delivering the correct values to the PEC. Each BRAM is paired with a PE, so BRAM$_0$ is connected to PE$_0$ and contains d_0, d_{0+1u}, d_{0+2u}, etc. Other BRAMs are initialized similarly, but with their values shifted downward.

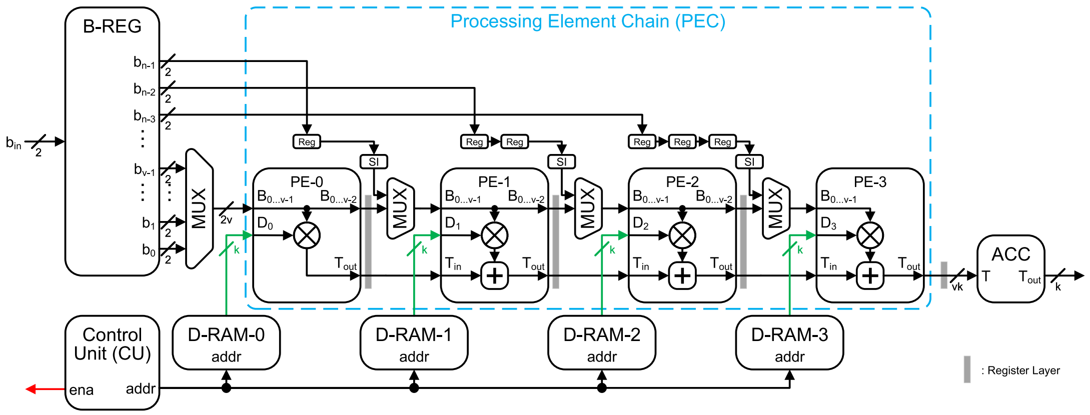
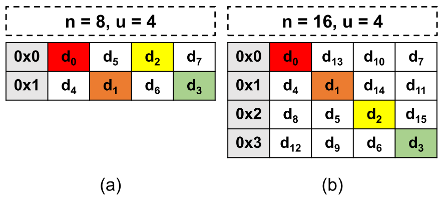


Fig. 1. Proposed ternary polynomial multiplication accelerator: SCALES ($u = 4$), where $B_{0\ldots v-1}$ denotes the needed v coefficients from $[B]$ for the related operations and D_0, D_1,..., D_3 represent the corresponding elements from $[D]$.



n = 8, u = 4

0x0	d_0	d_5	d_2	d_7
0x1	d_4	d_1	d_6	d_3

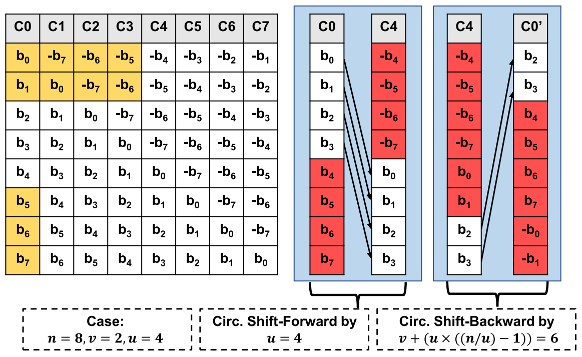
(a)

n = 16, u = 4

0x0	d_0	d_{13}	d_{10}	d_7
0x1	d_4	d_1	d_{14}	d_{11}
0x2	d_8	d_5	d_2	d_{15}
0x3	d_{12}	d_9	d_6	d_3

(b)

Fig. 2. D-RAM component arrangement (case example).



C0	C1	C2	C3	C4	C5	C6	C7
b_0	$-b_7$	$-b_6$	$-b_5$	$-b_4$	$-b_3$	$-b_2$	$-b_1$
b_1	b_0	$-b_7$	$-b_6$	$-b_5$	$-b_4$	$-b_3$	$-b_2$
b_2	b_1	b_0	$-b_7$	$-b_6$	$-b_5$	$-b_4$	$-b_3$
b_3	b_2	b_1	b_0	$-b_7$	$-b_6$	$-b_5$	$-b_4$
b_4	b_3	b_2	b_1	b_0	$-b_7$	$-b_6$	$-b_5$
b_5	b_4	b_3	b_2	b_1	b_0	$-b_7$	$-b_6$
b_6	b_5	b_4	b_3	b_2	b_1	b_0	$-b_7$
b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

C0	C4
b_0	$-b_4$
b_1	$-b_5$
b_2	$-b_6$
b_3	$-b_7$
b_4	b_0
b_5	b_1
b_6	b_2
b_7	b_3

C4	C0’
$-b_4$	b_2
$-b_5$	b_3
$-b_6$	b_4
$-b_7$	b_5
b_0	b_6
b_1	b_7
b_2	$-b_0$
b_3	$-b_1$

Case: $n = 8, v = 2, u = 4$
Circ. Shift-Forward by $u = 4$
Circ. Shift-Backward by $v + (u \times ((n/u) - 1)) = 6$

Fig. 3. Circulant matrix and B-REG shifting patterns.

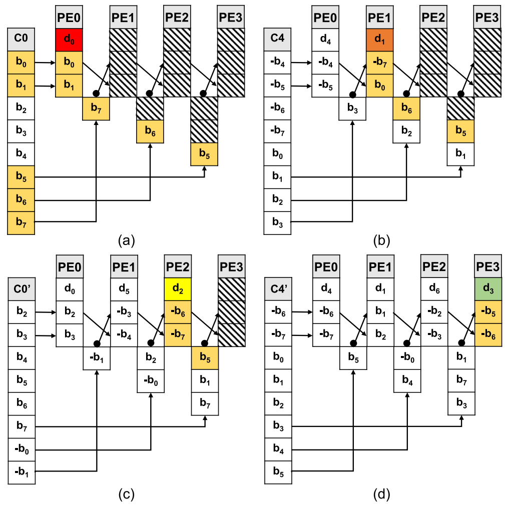


Fig. 4. Dataflow of the PEC (connecting Figs. 2 and 3).

Fig. 2 shows how values of D are being stored for $n = 8$ and $n = 16$, with each column representing a BRAM. Since the values are shifted downward, read addresses simply need to iterate from 0 to *depth*-1 repeatedly until the overall processing is complete.

B-REG: This component is an n-size 2-bit shift register for storing input ternary coefficients of B (b_i) and then producing correct signals to the PEC. During processing, there are two shifting functions needed to derive values of the circulant matrix: (i) Circular Shift-Forward by u and (ii) Circular Shift-Backward by $v + (u \times ((n/u) - 1))$, as shown in Fig. 3. During each cycle, the B-REG component is updated with a new shifted array of values to be fed to the PEC. Both shifting functions take one cycle (shown in blue boxes in Fig. 3, where red elements are being inverted).

Circular Shift-Forward. While processing through the entire circulant matrix $[B]$, we need to “jump ahead” to start processing a later $[B_{i,j}]$. To jump from column C0 to C4, we can circularly shift forward $u = 4$ times (Fig. 3).

Circular Shift-Backward. When pipelining for a given set of rows $[B_i]$ is complete, there is a need to “reset” the current column to C0 and shift upward to position the next rows. To perform this reset, we need to perform circular shift-backward $v + (u \times ((n/u) - 1))$ times. Fig. 3 demonstrates a reset from C4 to C0’ when $n = 8, v = 2, u = 4$. Note that b_2 and b_3 are now at the top, ready to be passed to the PEC.

PEC: The PEC performs the multiply-accumulate function of the overall matrix-vector multiplication process. Overall, a set of values pertaining to a $v \times u$ section of the circulant matrix can be passed to PE$_0$, and then the PEC is pipelined such that values propagate through without the need for additional input. On a given cycle during processing, each PE receives a v-size array of B values (multiplier), one D value (multiplicand), and a v-size array of T values to accumulate with (from previous PE). Each B value in the input array is multiplied with the input D value. Since B value is ternary $\{-1, 0, 1\}$, the result of each multiplication is one of {-D, 0, D}, which can be done with a MUX and is cheaper than larger DSP-based multipliers.

Fig. 4 shows the pipelining strategy within PEC, where a sample $v \times u$ section from Fig. 3 (yellow section) is connected. On cycle 0 (Fig. 4(a)), values from C0, namely b_0 and b_1, are multiplied with d_0 in PE$_0$, while b_5, b_6, and b_7 are passed to the

TABLE I
COMPARISON OF THE PERFORMANCE FOR THE EXIST DESIGNS AND THE PROPOSED DESIGN FOR THE FPGA IMPLEMENTATION

Design	Method	Device	q	LUT/FF/Slice/DSP/BRAM	Fmax (MHz)	CCs	Latency (μs)	Throughput (Mbps)	TPS[a]	ENS[b] (K)	eATP[c] (K)
n=1,024											
TVLSI'20 [5]	Iter. NTT	Virtex 7	2^{32}	77K/-/-/952/325.5	200.00	192	0.96	34.13K	183.17	186.35	178.89
	Four-Step			67K/-/-/599/129		280	1.40	23.41K	215.43	108.65	152.11
Access'22 [6]	Conv.	Pynq	2^{32}	37K/-/-/0/96	100.00	1,024	10.24	3.20K	266.67	30.84	315.75
TCAS-II'24 [2]	Conv.	Virtex 7	2^{30}	34.7K/33K/18.7K/0/2.5	178.57	1,024	5.73	5.36K	285.99	19.22	110.23
			2^{32}	36.8K/35K/19.3K/0/2.5				5.71K	296.12	19.79	113.47
Pro. ($v=32, u=4$)	Block Processing	Virtex 7	2^{32}	13.4K/8.6K/3.5K/0/2	303.95	8,196	26.95	1.22K	352.20	3.84	103.61
Pro. ($v=32, u=8$)				21.3K/13.1K/5.7K/0/4	277.01	4,096	14.79	2.21K	387.63	6.50	96.14
Pro. ($v=64, u=8$)				38.5K/23.8K/10.1K/0/4	257.73	2,048	7.95	4.12K	408.29	10.88	86.49
Pro. ($v=64, u=16$)				72.2K/41.5K/19.0K/0/7.5	229.89	1,024	4.45	7.36K	387.79	20.46	91.14
n=4,096											
MM'20 [3]	NTT	Virtex 7	2^{32}	39.6K.21.1K/-/224/96	150.00	5,686.5	37.91	3.46K	19.90	173.77	6.59K
TCAS-II'24 [2]	Conv.	Virtex 7	2^{32}	212K/139.5K/69.8K/0/8.5	131.58	4,096	31.13	4.21K	60.30	71.50	2.23K
Pro. ($v=16, u=16$)	Block Processing	Virtex 7	2^{32}	28.5K/18.6K/7.5K/0/7.5	281.69	65,535	232.65	563.38	74.82	9.02	2.10K
Pro. ($v=64, u=8$)				47.3K/30.0K/12.6K/0/4	234.74	32,768	139.59	938.97	74.58	13.37	1.87K
Pro. ($v=64, u=16$)				79.7K/47.7K/21.3K/0/7.5	219.78	16,384	74.55	1.76K	82.55	22.79	1.70K
Pro. ($v=128, u=16$)				147K/86.5K/39.3K/0/7.5	172.71	8,192	47.43	2.76K	70.25	40.83	1.94K

[a]: TPS=Throughput/Slice. [b]: BRAMs and DSPs are converted to equivalent Slice. ENS: equivalent number of slices. [c]: eATP=#ENS×Latency.

delay registers. On cycle 1 (Fig. 4(b)), C0 is shifted to C4 and D-RAM increments address to produce the next set of D values. Now, values from C4, namely $-b_4$ and $-b_5$ are multiplied with d_4 in PE$_0$, while b_1, b_2, and b_3 are passed to the registers. Similar operations can be seen in Fig. 4(c) and (d), which repeats until all rows are processed.

ACC: The ACC component accumulates PEC results of multiple sequential pipeline phases and shifts results out. While processing, it receives a v-size array of T values from PE$_{u-1}$ and sums with any previously accumulated values. When processing for a given set of rows $[B_i]$ is complete, the accumulated result T_i values (Algorithm 1) are ready to be delivered out, and this repeats until all T is delivered out.

Brief Summary of Key Techniques: Overall, we have: (i) arranged the small coefficient and related signal processing through 2-bit-size shift-register; (ii) minimized register usage in pipelined PEC with small coefficients, and delay register technique; (iii) implemented the point-wise multipliers with simple MUXes; (iv) handled large values by BRAMs to avoid register usage; (v) processed the computation in a systolic way; (vi) equipped the accelerator with scalability.

IV. IMPLEMENTATION AND COMPARISON

Experimental Setup: The experimental setup is as follows: (a) we have coded the accelerator with VHDL and verified its functionality with ModelSim; (b) we have selected polynomial size $n = 4,096/1,024$ and $q = 2^{32}$ ($k = \log_2 q =$ 32-bit). We implemented the design with many combinations of v and u and have presented the ones with the lowest eATP; (c) we have implemented the accelerator through Vivado 2020.2 on the Virtex-7 (XC7VX690TFFG1761-2) device, following [2]; (d) we have obtained the implementation results (after place & route) including the number of LUTs, FFs, slices, BRAMs, and maximum frequency; (e) we have followed the resource equivalency calculation of [2] to calculate the equivalent area-time product (eATP); (f) finally, results are presented in Table I for a comprehensive comparison.

Comparison: When comparing with the most recent [2], the proposed SCALES ($v = 64, u = 16$) involves 19.0% less eATP than [2] for $n = 4,096$. Further, for $n = 1,024$, SCALES ($v = 64, u = 8$) requires 23.8% smaller eATP compared to [2]. Though the accelerator of [2] is also a systolic structure, it does not offer scalability as the proposed one. Thus, we can conclude that the overall performance of the proposed design is better than [2].

When comparing with [5], [6], and [3], we want to emphasize that the authors of [2] have already shown their design's efficiency over the others. By extension, it is shown that our accelerator also has better area-time complexities than these ones.

V. CONCLUSION

This paper delivers a novel ternary polynomial multiplication accelerator (SCALES) for cryptographic applications like homomorphic encryption (BFV scheme). We have derived a new block-processing strategy and algorithm for ternary polynomial multiplication and implemented it as an FPGA-based systolic accelerator. Taking advantage of various design optimization strategies, SCALES offers efficient, flexible, and scalable processing, and the resulting implementation and comparison have confirmed the efficiency of the proposed accelerator over the state-of-the-art designs.

REFERENCES

[1] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptol. ePrint Arch.*, 2012.

[2] J. Ren et al., "An efficient ring polynomial multiplication accelerator for homomorphic encryption," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 71, no. 1, pp. 415–419, Jan. 2024.

[3] A. C. Mert, E. Öztürk, and E. Savaş, "FPGA implementation of a run-time configurable NTT-based polynomial multiplication hardware," *Microprocessors Microsystems*, vol. 78, 2020, Art. no. 103219.

[4] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Proc. Annu. Cryptol. Conf.*, Springer, 2012, pp. 868–886.

[5] A. C. Mert, E. Öztürk, and E. Savaş, "Design and implementation of encryption/decryption architectures for BFV homomorphic encryption scheme," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 2, pp. 353–362, Feb. 2020.

[6] I. Syafalni, G. Jonatan, N. Sutisna, R. Mulyawan, and T. Adiono, "Efficient homomorphic encryption accelerator with integrated PRNG using low-cost FPGA," *IEEE Access*, vol. 10, pp. 7753–7771, 2022.

[7] H.-T. Kung, "Why systolic architectures?," *Computer*, vol. 15, no. 1, pp. 37–46, 1982.

[8] J. Xie, C.-Y. Lee, P. K. Meher, and Z.-H. Mao, "Novel bit-parallel and digit-serial systolic finite field multipliers over $GF(2^m)$ based on reordered normal basis," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 9, pp. 2119–2130, Sep. 2019.

[9] J. M. Pollard, "The fast fourier transform in a finite field," *Math. Computation*, vol. 25, no. 114, pp. 365–374, 1971.

[10] A. Karatsuba, "Multiplication of multidigit numbers on automata," *Sov. Phys. Doklady*, vol. 7, pp. 595–596, 1963.