coxHE: A software hardware co-design framework for FPGA acceleration of homomorphic computation

Mingqin Han[a*], Yilan Zhu[a*], Qian Lou[b], Zimeng Zhou[a], Shanqing Guo[a], Lei Ju[a]
[a] School of Cyber Science and Technology, Shandong University, Qingdao, China
[b] Intelligent Systems Engineering department, Indiana University, Bloomington, Indiana
{sduhmq,yilanzhu}@mail.sdu.edu.cn, {louqian}@iu.edu,{zhouzimeng,guoshanqing,julei}@sdu.edu.cn

***Abstract*—Data privacy becomes a crucial concern in the AI and big data era. Fully homomorphic encryption(FHE) is a promising data privacy protection technique where the entire computation is performed on encrypted data. However, the dramatic increase of the computation workload restrains the usage of FHE for the real-world applications. In this paper, we propose an FPFA accelerator design framework for CKKS-based HE. While the key-switch operations are the primary performance bottleneck of FHE computation, we propose a low latency design of key-switch module with reduced intra-operation data dependency. Compared with the state-of-the-art FPGA based key-switch implementation that is based on Verilog, the proposed high-level synthesis (HLS) based design reduces the operation latency by 40%. Furthermore, we propose an automated design space exploration framework which generates optimal encryption parameters and accelerators for a given application kernel and the target FPGA device. Experimental results for a set of real HE application kernels on different FPGA devices show that our HLS-based flexible design framework produces substantially better accelerator design w.r.t. a fixed-parameter HE accelerator in terms of security, approximation error, and overall performance.**

***Index Terms*—Homomorphic encryption, FPGA acceleration, high-level synthesis, design space exploration**

I. Introduction

Data privacy becomes an increasing important concern in the AI and big data era. When user data are manipulated on the cloud servers, how to ensure data privacy is a now a critical trust gap between the data owner and the cloud service provider. Fully homomorphic encryption (FHE) is a very promising data privacy framework, where the entire computation on the server is operated on the encrypted data [1]. With the rapid growth in the past years, FHE has drawn great attention from both academia and industry. For instances, Microsoft has launched the open-source SEAL library for FHE [2]. CrytpoNets [3] made it possible to perform CNN inference task with FHE. Google has recently open-sourced a general-purpose "transpiler" able to convert high-level code to be used with Fully Homomorphic Encryption (FHE) [4].

Many different schemes has been proposed to realize FHE, such as BGV [5], BFV [6], TFHE [7], and CKKS [8]. However, one of the critical obstacles to deploy these schemes for real-world practice is the enormous computation overhead compared to directly process the plaintext data without FHE. In general, a single float or integer data would be encrypted into a series of polynomials involving thousands of coefficients, which makes the FHE schemes incurs 3-4 orders of magnitude higher computation and storage overhead. For instance, Cryptonets [3] requires 250 seconds to perform CNN inference with FHE, while the latency would be at millisecond level for doing the same task with plaintext.

As a result, design domain specific architecture and accelerator for FHE is of paramount importance for practical FHE-based applications. Dai et al. [9] proposed a GPU-based FHE library "cuHE" which utilizes the massive parallel processing model and high memory bandwidth of GPGPUs. In April 2021, Intel has released Homomorphic Encryption Acceleration Library (HEXL) [10] which uses the Intel Advanced Vector Extensions 512 (Intel AVX512) instruction set to enable high-performance multi-threaded execution of Microsoft SEAL on Intel processors. FPGA acceleration for basic operations of FHE has been studied in [11]–[13]. Recently in [14], Riazi et al. proposed FPGA-based hardware architecture for CKKS FHE, where the fundamental number-theoretic transform (NTT) blocks and high-level parallel operations can be scaled to different encryption parameters and hardware resources. In [15], multi-level parallelism design has been exploited to achieve better FPGA resource utilization for the Multiply-Accumulate operations in matrix-vector multiplication.

Existing FPGA accelerators focus on the hardware design of low- and/or high- level FHE operations (e.g., NTT, key-switching, re-scaling), based on the internal structure and data dependency of these operations. However, we claim that the application-level information are crucial for practical FHE kernel acceleration, because

- Given an FHE application, dozens- or hundreds- of encryption parameters combinations are available (e.g., related to the multiplication depth in the software kernel). The encryption parameter selection leads to trade-offs between the security level, hardware resource usage, and FHE approximation error.
- The software kernel determines the required FHE operation types, inter-operation dependency, as well as the operation frequency. In order to fully utilize the FPGA resource for an optimal accelerator design, the kernel-level information must be incorporated in the hardware core design and deployment.

In this paper, we propose a flexible FPGA co-design acceleration framework for CKKS-based FHE kernels (coxHE). For a given software FHE kernel and target FPGA device, the proposed coxHE framework automatically determines the encryption parameters as well as FHE operations hardware core design. To the best of our knowledge, this is the first FPGA

FHE accelerator that is entirely built with high-level synthesis (HLS) design methodology, which provides extreme flexible design and fast deployment across different FPGA devices. The contributions of this paper are summarized as follows:

- We remove data dependencies within the bottleneck KeySwitch operation to reduce its latency by up-to 48%, compared with the state-of-the-art FPGA CKKS accelerator. Meanwhile, the HLS-based implementation of the FHE operations are parametric to enable fast design space exploration between hardware resource usage and performance.
- The coxHE framework consider multiple design objectives including security level, performance, and approximation error. It automatically determines the Pareto optimal design solutions within the huge system design space composed by possible encryption parameters and hardware core design choices (e.g., parallel levels, data placement).
- While most literature work on FPGA FHE accelerator evaluate the design at FHE operation level (e.g., NTT, KeySwitch), coxHE framework contains the full-fledged design flow and is evaluated on a set of real-world FHE software kernels. Compared to Intel HEXL based on multi-threading and AVX512 on i7-8700@3.70GHz, the generator accelerator achieves up-to 42.7X energy efficiency on Xilinx ZCU102.

II. Background

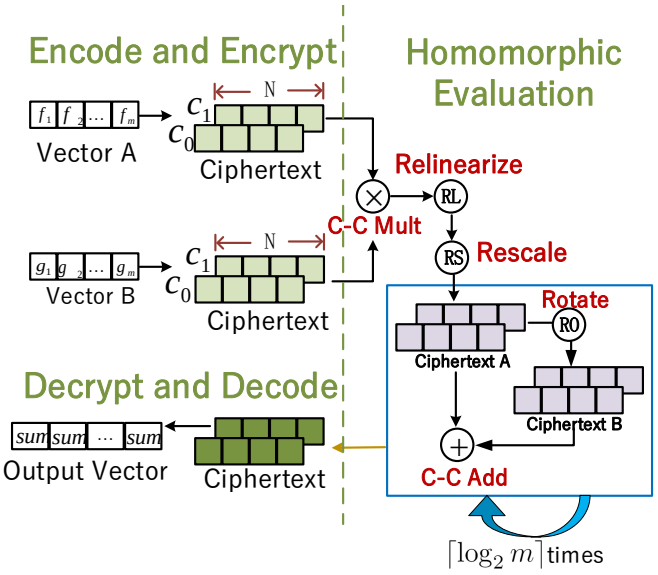


Fig. 1. Encrypted vector multiplication. The circled operations are high-level CKKS operations, and the dashed box is repeated $\lceil \log_2 m \rceil$ times.The output vector contains the results where $sum = \sum_{i=1}^{m} f_i \cdot g_i$.

The primary advantage of FHE is to enable computation on encrypted data without decrypt them. It is attributed to the homomorphism of its addition and multiplication which can be represented by the following formula.

$$\begin{aligned} HE.enc(a+b) &= HE.enc(a) + HE.enc(b) \\ HE.enc(a \cdot b) &= HE.enc(a) \cdot HE.enc(b) \end{aligned} \quad (1)$$

CKKS is a popular FHE scheme which is designed for arithmetic of approximate numbers to supports homomorphic evaluations on floating-point numbers. Fig. 1 illustrates the process of CKKS scheme for encrypted vector multiplication. The data is encoded and encrypted as a pair of ciphertext polynomials. Ciphertexts are then input as operands to homomorphic addition and multiplication operations at server side, and obtain the result ciphertexts. Finally, the result ciphertexts are decrypted and decoded to obtain the correct results at the user side. Compared with encode, encrypt, decrypt and decode operations, the homomorphic evaluation operations are generally the bottleneck of execution time and energy consumption for CKKS scheme. Therefore, the homomorphic evaluation optimization is the crucial for CKKS optimization.

A ciphertext is denoted as two polynomials $(\boldsymbol{c}_0, \boldsymbol{c}_1)$, where $\boldsymbol{c}_k = \sum_{i=0}^{N-1} a_i x^i \pmod{Q}$, $k \in (0,1)$. N is usually selected as 2^{10}, 2^{11} ... 2^{14}, 2^{15}, and the bit-width of Q is up to hundreds of bits. The computationally intensive modular arithmetic on big integers cause the dramatic performance degradation for the FHE computation. Fortunately, the RNS variant of CKKS [16] could divide Q into small q_i, $Q_l = \prod_{i=0}^{l} q_i$ and make high parallelism possible in FPGA.

To perform homomorphic addition (HADD) and multiplication (HMULT), CKKS scheme provide support for plaintext-ciphertext (P-C) operations, ciphertext-ciphertext (C-C) operations, rescale, relinearization and rotation. Among these operations, P-C and C-C operations are basic and simple to implement. Relinearize is used after C-C Mult to prevent the expansion of the number of ciphertext polynomials. It is worth mentioning that CKKS supports SIMD-style technique that it can encode a vector into a plaintext and then encrypt it. That is, the same operation can be performed on each element of the vector with only a homomorphic operation. Rotate operation can change the order of the vector elements in one direction when it encrypt. Then we can make a sum of all the elements in a vector through consecutive rotation and C-C Add. It is easy calculate the number of consecutive rotations as $\lceil \log_2 m \rceil$ shown in blue box in Fig. 1. Therefore, this feature is often used in computing homomorphic vector multiplication and matrix multiplication if the elements packed in a ciphertext. Besides, Rescale can be used to reduce the number of small modulus q_i and avoid the underlying plaintext values in the ciphertext to blow-up.

III. Motivation

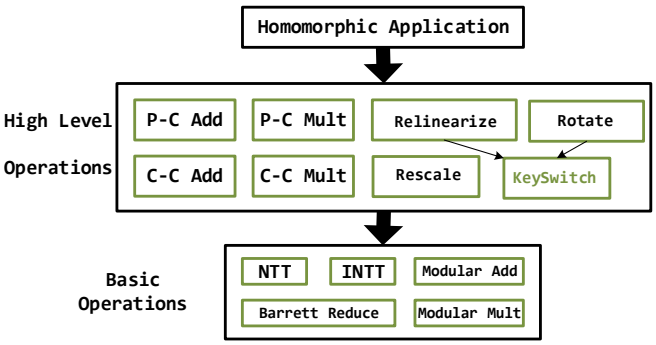


Fig. 2. The hierarchy of homomorphic operations.

KeySwitch Acceleration. Fig. 2 shows the hierarchy of various CKKS homomorphic operations. Among the high-level

operations, Relinearize and Rescale both invoke the KeySwitch operation, with some additional pre-/post- operations. Generally, KeySwitch is the most computationally intensive and time-consuming operation in FHE. For example, the running time of one single KeySwitch invocation is almost 10 times that of P-C multiplication, 7 times of C-C multiplication, 2 to 3 times of Rescale. The total time consumed by KeySwitch operations is even as high as 90% for homomorphic matrix multiplication. Therefore, design efficient KeySwitch accelerator is crucial for FHE accelerator on FPGA.

HEAX [14] has designed a fine-grained pipeline architecture for the KeySwitch operation, which achieves optimal throughput when the pipeline is perfectly filled. However, in this paper, we argue that attention should also be paid to the latency optimization.

As the encrypted vector multiplcation example shown in Fig. 1, the KeySwitch operation (invoked by the Rotate operation) in the dashed box will be executed $\lceil \log_2 m \rceil$ times. However, due to the data dependency exists between consecutive loop iterations, the execution of the KeySwitch operation cannot be pipelined. Therefore, the latency (rather than the pipelined throughput) is the key optimization goal in this scenario. And this is a very common case in typical CKKS kernels.

We evaluate the effectiveness of coxHE design space exploration with the plain-cipher matrix multiplication benchmark.

Design space exploration for multi-objective optimization. Existing work on FPGA acceleration of FHE focus on the hardware design of FHE low- and/or high-level operations to improve the performance of FHE. However, a practical CKKS kernel accelerator typically has multiple design objective including security, performance (energy efficiency), and approximation error (computation accuracy). Given a software kernel, hundreds combinations of encryption parameters (i.e., N and q_i) are possible, each has potentially different impact on the design objectives. Meanwhile, for a given encryption parameters selection, we have many possible hardware configurations for each CKKS low- and high-level operations (e.g., intra-parallelism, data layout) on a particular FPGA device. Therefore, the accelerator designer has a huge design space where exhaustive exploration is impossible.

TABLE I
TWO POSSIBLE DESIGN DESIGN CHOICES.

	N	q_i	security	latency(ms)	error
S1	8192	33	192	34.56	9.53×10^{-6}
S2	8192	37	192	67.34	9.53×10^{-7}

For instance, assume we accelerate a CKKS kernel for plain-cipher matrix multiplication on Xilinx ZCU104 device. If we set the encryption parameter N and q_i bit-width to be 16384 and 40, it requires at least 47.19Mbit on-chip memory, which is larger than the total available on-chip memory of ZCU104 and thus not a feasible encryption parameter selection. On the other hand, Table I shows two possible design solutions S1 and S2. The encryption parameters selected for S2 lead to a KeySwitch accelerator with lower internal parallelism due to hardware resource constraints. Therefore, considering the same security level and similar approximation error (between 10^{-6} and 10^{-7}), S1 could be a better design choice for most circumstances. In the proposed framework, we perform automatic design space exploration to generate Pareto optimal solutions with trade-offs among different design objectives, out of the huge design space due to encryption parameter selection and possible hardware configurations.

IV. ACCELERATOR ARCHITECTURE

In this section, we first introduce our low-latency accelerator design of the KeySwitch operation in Section IV-A. In Section IV-B, we present the parametric design of CKKS operations based on the high-level synthesis (HLS) methodology ([17]), which enables automatic design space exploration for the system-level software hardware co-design.

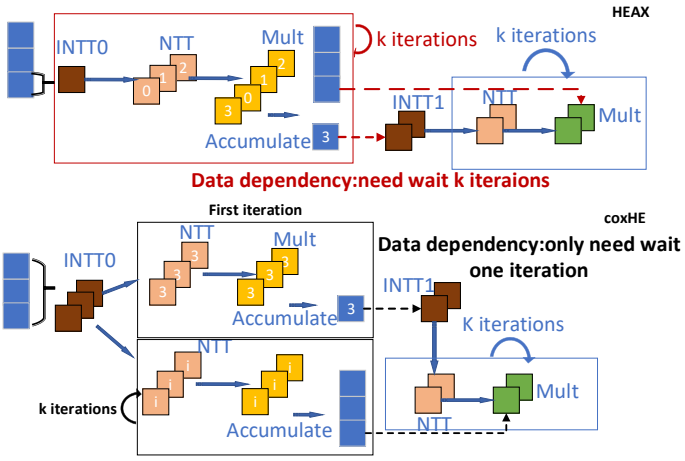


Fig. 3. Elimination of data dependency in KeySwitch.

A. Low-latency KeySwitch acceleration

As we have discussed in SectionIII, KeySwitch is highly optimized in the proposed coxHE framework to obtain reduced latency. As shown in the Fig. 3, the coxHE eliminates two data dependencies of KeySwitch to reduce the execution time compared with HEAX [14]. In HEAX, the INTT1 module needs to wait k iterations of the former layer which brings long halt for the subsequent modules. We find that the INTT1 only needs the output of the input polynomials computing with the last modulus, while each iteration consumes all moduli in HEAX. Then we eliminate this unnecessary dependency so that the KeySwitch operation achieves lower latency. Specifically, we reorganize the computing order of the moduli and their corresponding polynomials in the former layer while keeping the same whole computing amounts. coxHE firstly computes the last modulus within the input polynomials which is essential and enough for INTT1. Thus, INTT1 only need wait one iteration of the former layer to start its computing task, so the overall latency would be reduced a lot. And the latency reduction become significant for a larger number of moduli which is correlated to the encrytpyion parameter selection.

B. HLS-based accelerator design

Listing 1. Coding style

```
typedef ap_uint<BITWIDTH> UDTYPE;
template<unsigned mod_count>
```

```
void P-C_Mult(...)    {
#pragma HLS INLINE off
    ...
    for (unsigned i = 0; i < 2; i++)
        for (unsigned j = 0; j <
            coeff_mod_count; j++)
        #pragma HLS UNROLLL factor=4
            Modular_Mult(...);
}
```

HLS provides the inbuilt directives allowing us to reconfigure our design without changing significantly the original code. Our coxHE framework is designed to support polynomials of any degree and any coefficients size. The coxHE also supports to change the coarse- or fine- grained of parallelism among both high level modules and basic operations listed in Fig. 2. Taking the P-C Mult operation as an example, we can adjust its fine-grained parallelism by adding the UNROLL directive and its factor for the corresponding loops to achieve design tradeoffs between resource usage and performance.

To change the bit-width of the q_i, we can use ap_uint data type. As for data arrangement, coxHE can set various memory layout for better optimization. We also use the template coding style which makes our code flexible to reconfigure. The flexibility of coxHE makes it possible to explore the wide design space towards various application kernels.

V. Design Space Exploration

As we have discussed in Section III, it is a non-trivak task for a system developer to choose the best design choice defined by the encryption parameters as well as hardware structure for variable CKKS operations. In this section, we formulate the multi-objective optimization problem by modeling the performance, resource utilization and the approximation error for each basic module and use them for fast design space exploration. Finally, we solve the optimization problem to generate a set of HE parameters configurations that can be used by the HLS-based FPGA acceleration design framework to automatically generate a set of Pareto optimal accelerators.

A. Problem formulation

The resource-constrained performance optimization problem can be formulated as follows:

$$
\begin{aligned}
&Min: \Sigma_0^m L(modules)\\
&Min: E\\
&Max: S\\
&Subject\ to\\
&\qquad Mem_{KeySwitch} + Mem_{top} \le Mem_{Max}\\
&\qquad \Sigma_{i=0}^{Num_{bm}} DSP_i \le DSP_{max}\\
&\qquad \Sigma_{i=0}^{Num_{bm}} Logic_i \le Logic_{max}\\
&\qquad \Sigma_{i=0}^{k} bitwidthp_i = \sigma\\
&\qquad E \le error_{max}, \quad S \ge security_{min}
\end{aligned}
$$

where L denote the execution time of the kernel, E represents the approximation error and S denotes the security level. The $bitwidth$ means the bit width of the data.$DSP_{max}, Memory_{Max}$ and $Logic_{max}$ represent the total DSP, on-chip memory and FPGA logic resources,respectively, available in a given FPGA.

B. Performance model

As we mentioned before, the basic modules consist of Mod_Add (Modular Add), Mod_Mult (Modular Mult), Barrett_Reduce and NTT/INTT which can be orgnized to C-C Mult, C-C Add, KeySwitch, escale module and so on. The execution time of these modules is analytically modeled as several functions of design variables and validated by performing the design points and running them on the FPGA accelerator.

1) basic modules executing time: The inputs of the NTT/INTT are all existed on on-chip BRAM, recalling our NTT/INTT's compute flow, thus we can easily get the execution time of NTT/INTT:

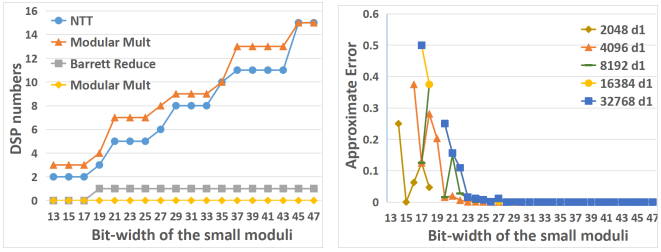
$L_{NTT} = logN * (N/2/PIntra_{NTT})$, where $PIntra_{module}$ represents the number of cores within one module, For other basic modules, they all process the data sequentially. We first consider the case that all the input data is on the BRAM. The relationship between the Mod_Mult module's intra parallel degree and its latency can be expressed as follows:

$L_{Mult} = N/PIntra_{Mult}$.

For those Mod_Mult modules that belong to keyswitch module,they need compute the Mod_Mult product between the output of NTT modules and the relinearization/Galois keys. Because the keys are too large to be stored on BRAM, we keep them on DDR. So these Mod_Mult modules' performance is also limited by the memory bandwidth. Notice that the access pattern of the relinearization/Galois keys is sequential, thus we can use burst mode to enable efficient data transfer from external memory. To take full advantage of the bandwidth, we package multiple Relinearization /Galois keys together. We define the number of keys packed on one port as $packnum$ and the number of ports we actually used in a high level module as $portnum$. They should satisfy the following relationships to pipeline the basic modules in KeySwitch:

$PIntra_{Mult} = packnum * portnum/4$.

C. Resource model



(a) Resource model. (b) Error model.

Fig. 4. The DSP and error for varied bit-width in basic modules.

It is not feasible to analytically model the FPGA computing and logical resource utilization of the kernel implemented by HLS because of the optimizations performed in the HLS tools. Therefore, we use the synthesis results to emprically model the FPGA resource utilization. We present the DSP, FF and LUT resource usage of the basic modules in different bitwidths of the small modulus q_i while setting the intra parallel degree

to 1 as fig.4(a). As we can see, for NTT/INTT, Mod_Mult and barrett_Reduce, the DSP functional relationship presents a ladder-like overall upward trend, which means that different bitwidths may correspond to the same DSP usage. However, for a given bitwidth range, the approximation error with different bitwidth varies from each other which is shown in 4(b). For all the basic modules,the amount of FF and LUT increases roughly with the bitwidth grows.

D. Error model

This part involves the influence of homomorphic parameter selection on FHE kernels, which belongs to the software level of our DSE. The parameters for us to choose are N and q_i of the formula $Q_L = \sum_{i=0}^{L} q_i$, which mainly impact the security level and the accuracy of results after decrypted. To explore the design space, we model the accuracy for FHE kernels which called approximate error exactly.

In CKKS [8], there is a scale factor to maintain the accuracy of floating point numbers when performing homomorphic evaluation, which is denoted as q. According to [18] and [16], the choice of q_i should be as close to q as possible. It is denoted that $q/q_i \in (1-2^{-\epsilon_i}, 1+2^{-\epsilon_i})$. To calculate the error caused by a homomorphic multiplication, two plaintexts denote by $q \cdot \mu_1$ and $q \cdot \mu_2$. The result after a multiplication and a rescale is denoted as $r_R = \frac{q^2}{q_i} \cdot \mu_1 \cdot \mu_2$, and the orginal result without error is denoted as $r_C = q \cdot \mu_1 \cdot \mu_2$. The error is represented in the following formula:

$$\begin{aligned} E_{depth=1} &= \frac{|r_R - r_C|}{r_C} \\ &= \frac{|\frac{q^2}{q_i} \cdot \mu 1 \cdot \mu 2 - q \cdot \mu 1 \cdot \mu 2|}{q \cdot \mu 1 \cdot \mu 2} \\ &\leq \frac{2^{-\epsilon_i} \cdot q \cdot \mu 1 \cdot \mu 2}{q \cdot \mu 1 \cdot \mu 2} = 2^{-\epsilon_i}. \end{aligned} \tag{2}$$

The approximation error with multiplication depth d is:

$$E_{depth=d} = \sum_{l=0}^{d-1} 2^{-\epsilon_{i-l}+(d-1-l)}. \tag{3}$$

Fig. 4(b) shows the relationship between the approximate error ratio and the bit-width of q_i for different N when multiplication depth is 1. Due to the impact of security on bit-width of Q, not all the bit-width can be set for each N. For example, when $N = 2048$, it can only select the number of bits between 14 and 18. For the same bit-width of q_i, the smaller N means the smaller error in general. However, if high accuracy and security are required, a larger N could be set to achieve it. It seems that when the bit-width is greater than 27, its error will be close to zero. But it caused by the 1 depth of multiplication. Formula (4) shows the error produced by multiple consecutive multiplication, which will increase greatly with the multiplication depth grows.

VI. Experiments

A. Performance comparison with HEAX

We first evaluate the performance of the bottleneck KeySwitch operation w.r.t. the state-of-the-art FPGA CKKS accelerator HEAX ([14]). The pipelined KeySwitch design of HEAX targets to maximize it throughput. Our proposed design archives the same throughput with equal slowest pipeline stage. On the other hand, our design reduces the latency of the KeySwitch operation, which benefits the overall performance when the pipeline is not fully filled in common cases.

As shown in Fig. 5(a), for different number of moduli, we reduce the KeySwitch operation latency by 33%-48% compared with HEAX. While HEAX only reports the performance of individual operations, we extend the HEAX framework and compare the kernel-level performance under the same settings (i.e., $Pintra_{NTT/INTT} = 16$, $Pintra_{Dyadic} = 8$). As shown in Fig. 5(b), the proposed framework achieve 11.4%-25.77% overall performance improvement with the same simple deployment (i.e., without design space exploration).

TABLE II
THE SETTINGS OF BENCHMARKS.

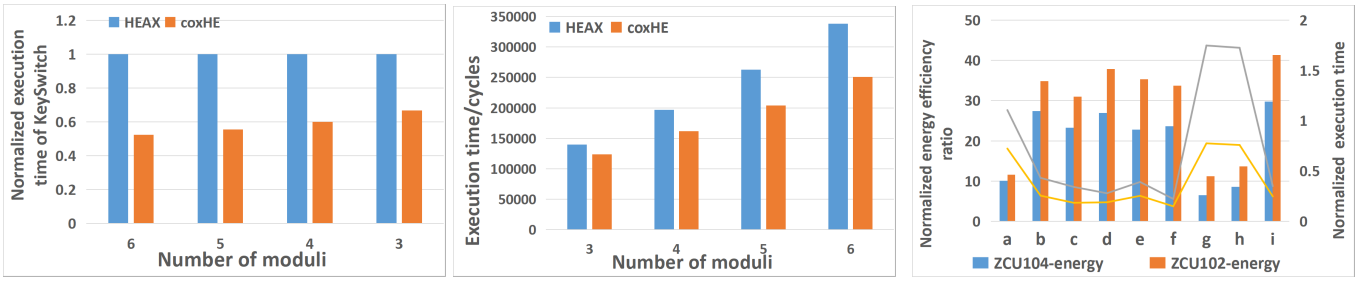
	Benchmark	Operation	Size
a	DotPlainBatchAxis_CKKS	Plain-Cipher	(100x10)x(10x1)
b	DotCipherBatchAxis_CKKS	Cipher-Cipher	(100x10)x(10x1)
c	MatMultVal_CKKS	Plain-Cipher	(1x2048)x(2048x1)
d	MatMultVal_CKKS	Plain-Cipher	(1x10)x(10x1)
e	MatMultVal_CKKS	Cipher-Cipher	(1x2048)x(2048x1)
f	MatMultVal_CKKS	Cipher-Cipher	(1x10)x(10x1)
g	MatMultVal_CKKS	Plain-Cipher	(100x10)x(10x1)
h	MatMultVal_CKKS	Cipher-Cipher	(100x10)x(10x1)
i	Average_pool	Cipher-Cipher	(20x20)x(4x4)

B. Performance comparison with HEXL

The Intel Homomorphic Encryption Acceleration Library (HEXL) supports Microsoft SEAL CKKS library with Intel AVX-512 acceleration instructions. HEXL achieves 1.23X-5.46X speedup for various CKKS operations compared with SEAL for single-threaded execution, and the performance gain is scalable with number of CPU cores for multi-threaded execution. In our evaluation, HEXL is execution with 12 threads on Intel Core i7-8700@3.70GHz (6-core, 12-thread, 95W TDP, 50-90W measured runtime power). The proposed coxHE framework generates CKKS accelerator on Xilinx Zynq UltraScale+ MPSoC ZCU102 (600K logic cells, 32.1Mbit BRAM, 2520 DSP slices, 6-8W measured runtime power) and ZCU104 (504K logic cells, 38Mbit BRAM+URAM, 1728 DSP slices, 5-7W measured runtime power). We use the SEAL Sample Kernel benchmark plus an "average pooling" kernel to evaluation the performance and energy efficiency of HELX and coxHE (Table II). The evaluation results shown in Fig. 5(c) indicates that coxHE achieves reasonable performance speedup and substantial energy efficiency improvement on embedded FPGA devices (5-8W power) w.r.t. HEXL on desktop CPUs (95W TDP).

C. Design space exploration

The proposed coxHE automatically generates the Pareto optimal solutions for a given software kernel and target FPGA device. We evaluate the effectiveness of coxHE design space exploration with the plain-cipher matrix multiplication kernel, where the Pareto optimal design choices (out of thousands of possible solutions) are listed in Fig. III. For example, when



(a) KeySwitch performance. (b) C(1x10)*P(10x1) execution time. (c) Normalized energy efficiency and performance.

Fig. 5. Performance comparison of the proposed framework with HEXL.

TABLE III

CONFIGURATION AND PERFORMANCE OF OPTIMAL CHOICES SAMPLE USING OUR DSE METHOD.

	Homomorphic parameters		hardware configuration				%Utilization					Performance			
	N	qi	board	p-intra	portnum	packnum	DSP	FF	LUT	BRAM	URAM	error	security	Latency(ms)	Power(W)
A1	2048	18	ZCU104	8	1	2	79	14	43	100	17	4.68×10^{-2}	128	7.5008	6.035
A2			ZCU102	16	2	2	64	11	31	47				5.63	5.857
B1	4096	25	ZCU104	8	1	2	51	12	32	100	17	4.88×10^{-4}	192	15.1632	5.355
B2			ZCU102	16	2	2	69	17	49	66				9.3292	7.498
C1		33	ZCU104	8	1	2	40	8	21	61	17	9.65×10^{-6}	128	15.127	4.684
C2			ZCU102	16	2	2	95	23	37	83				8.17	6.451
D1	8192	25	ZCU104	8	1	2	50	13	30	88	92	4.88×10^{-4}	256	33.9432	5.751
D2			ZCU102	16	2	2	74	18	54	80				17.5857	7.909
E1		33	ZCU104	8	1	2	78	16	40	88	92	9.65×10^{-6}	256	34.5601	6.444
E2			ZCU102	16	2	2	96	21	56	70				20.7656	7.773
F1		43	ZCU104	4	1	1	41	11	28	69	58	1.86×10^{-8}	192	67.3475	5.368
F2			ZCU102	16	2	2	100	29	58	88				20.8008	7.854

$N = 8192$, there are three optimal configurations for ZCU104, i.e., D1, E1, and F1. The error of F1 is minimal while its latency is the longest one due to the bit-width of moduli is 43bit which leads to low intra-operation parallelism due to resource limitation. While both D1 and E1 have higher security level w.r.t. F1, D1 is the most energy-efficiency choice among these 3 solutions if the approximation error is tolerable. On the other hand, E1 is a preferable choice for application with very high accuracy demand, and it is only 2% slower than D1 (but requires more hardware resources).

VII. CONCLUDING REMARKS

In this paper, we propose an FPGA acceleration framework for CKKS based FHE. The full-fledged framework are built with high-level synthesis design flow, which generates FPGA accelerator for a given FHE software kernel and a target FPGA device. The proposed framework automatically explores the huge design space including encryption parameters and hardware configurations, and produce Pareto optimal solutions that balance security, performance, and approximation error.

REFERENCES

[1] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 79:1–79:35, 2018.

[2] H. Chen, K. Laine, and R. Player, "Simple encrypted arithmetic library - SEAL v2.1," *IACR Cryptol. ePrint Arch.*, p. 224, 2017.

[3] R. Gilad-Bachrach, N. Dowlin, K. Laine, and et al., "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *ICML, USA*, vol. 48. JMLR.org, 2016, pp. 201–210.

[4] S. Gorantala, R. Springer, S. Purser-Haskell, and et al., "A general purpose transpiler for fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, p. 811, 2021.

[5] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," in *Innovations in Theoretical Computer Science, USA*, S. Goldwasser, Ed. ACM, 2012, pp. 309–325.

[6] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, p. 144, 2012.

[7] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *ASIACRYPT*, vol. 10031, 2016, pp. 3–33.

[8] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *ASIACRYPT, Hong Kong, China*, vol. 10624. Springer, 2017, pp. 409–437.

[9] W. Dai and B. Sunar, "cuHE: A homomorphic encryption accelerator library," in *Second International Conference on Cryptography and Information Security in the Balkans*, vol. 9540. Springer, 2015, pp. 169–186.

[10] F. Boemer, S. Kim, G. Seifu, F. D. M. de Souza, and V. Gopal, "Intel HEXL: accelerating homomorphic encryption with intel AVX512-IFMA52," *CoRR*, vol. abs/2103.16400, 2021.

[11] Y. Doröz, E. Öztürk, and B. Sunar, "A million-bit multiplier architecture for fully homomorphic encryption," *Microprocess. Microsystems*, vol. 38, no. 8, pp. 766–775, 2014.

[12] D. B. Cousins, K. Rohloff, and D. Sumorok, "Designing an fpga-accelerated homomorphic encryption co-processor," *IEEE Trans. Emerg. Top. Comput.*, vol. 5, no. 2, pp. 193–206, 2017.

[13] A. Mkhinini, P. Maistri, R. Leveugle, and R. Tourki, "HLS design of a hardware accelerator for homomorphic encryption," in *DDECS, Germany*, M. Dietrich and O. Novák, Eds. IEEE, 2017, pp. 178–183.

[14] M. S. Riazi, K. Laine, B. Pelton, and W. Dai, "HEAX: an architecture for computing on encrypted data," in *ASPLOS, Switzerland*, J. R. Larus, L. Ceze, and K. Strauss, Eds. ACM, 2020, pp. 1295–1309.

[15] G. Xin, Y. Zhao, and J. Han, "A multi-layer parallel hardware architecture for homomorphic computation in machine learning," in *ISCAS, South Korea*. IEEE, 2021, pp. 1–5.

[16] J. H. Cheon, K. Han, A. Kim, and et al., "A full RNS variant of approximate homomorphic encryption," in *SAC, Canada*, ser. Lecture Notes in Computer Science, vol. 11349. Springer, 2018, pp. 347–368.

[17] J. Zhao, L. Feng, S. Sinha, W. Zhang, Y. Liang, and B. He, "COMBA: A comprehensive model-based analysis framework for high level synthesis of real applications," in *2017 IEEE/ACM International Conference on Computer-Aided Design, ICCAD*, 2017, pp. 430–437.

[18] A. Kim, A. Papadimitriou, and Y. Polyakov, "Approximate homomorphic encryption with reduced approximation error," *IACR Cryptol. ePrint Arch.*, p. 1118, 2020.