PQC Cloudization: Rapid Prototyping of Scalable NTT/INTT Architecture to Accelerate Kyber

Mojtaba Bisheh-Niasar
Microsoft
mojtabab@microsoft.com

Daniel Lo
Microsoft
dlo@microsoft.com

Anjana Parthasarathy
Microsoft
anjpar@microsoft.com

Blake Pelton
Microsoft
blakep@microsoft.com

Bharat Pillilli
Microsoft
bharat.pillilli@microsoft.com

Bryan Kelly
Microsoft
bryankel@microsoft.com

***Abstract*—The advent of quantum computers poses a serious challenge to the security of cloud infrastructures and services, as they can potentially break the existing public-key cryptosystems, such as Rivest–Shamir–Adleman (RSA) and Elliptic Curve Cryptography (ECC). Even though the gap between today's quantum computers and the threats they pose to current public-key cryptography is large, the cloud landscape should act proactively and initiate the transition to the post-quantum era as early as possible. To comply with that, the U.S. government issued a National Security Memorandum in May 2022 that mandated federal agencies to migrate to post-quantum cryptosystems (PQC) by 2035. To ensure the long-term security of cloud computing, it is imperative to develop and deploy PQC resistant to quantum attacks. A promising class of post-quantum cryptosystems is based on lattice problems, which require polynomial arithmetic. In this paper, we propose and implement a scalable number-theoretic transform (NTT) architecture that significantly enhances the performance of polynomial multiplication. Our proposed design exploits multi-levels of parallelism to accelerate the NTT computation on reconfigurable hardware. We use the high-level synthesis (HLS) method to implement our design, which allows us to describe the NTT algorithm in a high-level language and automatically generate optimized hardware code. HLS facilitates rapid prototyping and enables us to explore different design spaces and trade-offs on the hardware platforms. Our experimental results show that our design achieves 11$\times$ speedup compared to the state-of-the-art requiring only 14 clock cycles for an NTT computation over a polynomial of degree 256. To demonstrate the applicability of our design, we also present a coprocessor architecture for Kyber, a key encapsulation mechanism (KEM) chosen by the NIST post-quantum standardization process, that utilizes our scalable NTT core.**

***Index Terms*—cloud computing, cryptography, Kyber, NTT, post-quantum cryptography**

I. Introduction

Cloud computing has become an indispensable part of modern society, offering various services and applications to individuals and organizations. However, the security of cloud computing is threatened by the advent of quantum computers, which can potentially break the existing public-key cryptosystems, such as Rivest–Shamir–Adleman (RSA) and Elliptic Curve Cryptography (ECC) based on Shor's algorithm [1]. Current public-key cryptography is far from being threatened by today's quantum computers, but the cloud landscape should anticipate this challenge and initiate the transition to the post-quantum era in a timely manner. In alignment with this goal, the U.S. government issued a National Security Memorandum in May 2022 that mandated federal agencies to migrate to post-quantum cryptosystems (PQC) by 2035 to mitigate risks to vulnerable cryptographic systems [2].

The long-term security of cloud computing against quantum attacks depends on developing lattice-based cryptosystems, which are among the most promising PQC algorithms that are believed to be hard for both classical and quantum computers. The American National Institute of Standards and Technology (NIST) recognized this and selected CRYSTALS-KYBER and CRYSTALS-Dilithium, two lattice-based algorithms, as standards for post-quantum key-establishment and digital signatures, respectively, in July 2022 [3]. Lattice-based cryptography uses polynomial operations over a polynomial ring, which can be implemented efficiently using number theoretic transform (NTT) and inverse number theoretic transform (INTT). These transforms can greatly reduce the computational complexity of polynomial multiplication. NTT-based multiplication, which has a long history of use in various applications, especially in signal processing, is also a performance challenge for lattice-based cryptography implementation. Hence, many works have attempted to optimize NTT from different aspects, such as resource utilization, performance, efficiency, and energy consumption.

CRYSTALS-Kyber, a key encapsulation mechanism (KEM), is based on module learning-with-errors problem (M-LWE) in module lattices [4]. Kyber is notable for high-speed and constant-time implementations. As the next generation of the cryptosystem, Kyber is required to be implemented and evaluated on various platforms and applications, especially for cloud computing, which demands high performance and security. However, Kyber has not received enough attention in the cloudization framework. Therefore, exploring the hardware design of Kyber is necessary to exploit the advantages of FPGA-based architectures, such as parallelism, which can improve the system performance.

Hardware accelerators can be designed using two main approaches. RTL uses low-level languages such as VHDL or

Verilog to design a hardware architecture, which can offer more control and optimization, but it requires a longer time and a hand-optimized design that may sacrifice flexibility. On the other hand, high-level synthesis (HLS) uses high-level languages, which can offer flexibility and a shorter design cycle, but it may not achieve the best efficiency.

In this paper, we use the HLS approach to implement a pure hardware design of NTT architecture over the cloud, which can be faster and more flexible than other methods. HLS allows us to design a hardware architecture using high-level specifications, which can be mapped to FPGA and ASIC platforms with some optimizations. HLS also enables us to leverage the cloud resources to provide a scalable and secure environment for fast deploying a high-performance Kyber architecture.

A. Related Work

A flexible and scalable NTT architecture is presented in [5] targeting homomorphic encryption applications and post-quantum digital signatures. Their design can be tuned for different arithmetic configurations and the number of processing elements. Mert et. al. [6] present a flexible design for the NTT using an HLS design approach targeted on a Virtex-7 FPGA synthesized by Xilinx Vivado HLS tool. The paper reports that the HLS design requires on average 4.4 times more resources and 22.5 times more latency than the RTL design.

There are prominent works to design multi-core NTT architecture in the literature. Xing et. al. [7] propose an architecture utilizing 4 butterfly cores running in parallel on a Zync-7000 platform. Their design requires 2,688 cycles to perform an NTT with the NewHope parameter set. A 2×2 butterfly configuration is presented in [8], [9], and [10]. Between these works, Bisheh-Niasar et. al. [10] achieve a high-speed NTT architecture requiring 1,591 cycles for a polynomial NTT of degree 1,024 on an Artix-7 FPGA by merging NTT layers. This architecture requires 324 cycles for the Kyber parameter set with a polynomial of degree 256. In [11], a 3-layer merged NTT algorithm for the NewHope parameter set is implemented using RISC-V ISA features. However, the authors argue that applying this algorithm to Kyber would not yield any efficiency gains.

The pure hardware architectures of Kyber are proposed in [9], [10], [12]–[16]. One of the first initiatives of post-quantum acceleration using HLS was [12]. Furthermore, Nguyen et al. [9] compare the HLS- and RTL-based design methodologies for NTT and Kyber. They integrate the HLS implementation into the Xilinx SDSoC environment, showing that an HLS implementation obtained by modeling a block diagram is typically much better than an implementation obtained by using design space exploration. An instruction-set accelerator for Kyber is presented in [13] to design a flexible hardware architecture using a set of customized high-level instruction codes. The authors in [15] present a polynomial-vector multiplication unit taking advantage of polynomial vector structure in the Kyber. A resource-efficient modular reduction algorithm for Kyber is also proposed in [16].

B. Our Contributions

To the best of our knowledge, this work is the first hardware implementation of Kyber that focus on the cloud platform. Our work addresses performance, complexity, and design period challenges by proposing a novel framework for PQC cloudization. Our framework aims to design and implement a scalable and highly parallel framework based on NTT/INTT that can accelerate Kyber and other NTT/INTT-based PQC algorithms in cloud infrastructures and services. Our framework consists of a modular and flexible architecture that can support different NTT/INTT configurations. Our results show that our proposed framework using various optimization techniques, including multi-levels of parallelism, designing reconfigurable cores, and implementing interleaved and pipelined architecture, can achieve up to 11× speedup compared to existing NTT architectures while maintaining high security and scalability. Our proposed framework can facilitate the adoption and deployment of PQC in cloud computing, and enhance the security and efficiency of cloud services and applications in the post-quantum era.

The rest of this paper is organized as follows: Section 2 provides some background information on NTT-based polynomial multiplication and its importance for PQC algorithms such as Kyber. Section 3 describes our proposed PQC cloudization framework based on NTT/INTT in detail. Section 4 presents our experimental results and analysis on an FPGA device and cloud platform. Section 5 concludes the paper and suggests some future directions for our research.

II. PRELIMINARIES

A. Notations

Let q be a prime number and $\mathbb{Z}_q$ be the ring of integers modulo q. We define the ring of polynomials for some integer N as $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N+1)$, where the polynomials have n coefficients, each modulo q. We use regular font lowercase letters (a) to represent single polynomials, bold lowercase letters ($\mathbf{a}$) to represent polynomial vectors, and bold uppercase letters ($\mathbf{A}$) to represent a matrix of polynomials. Besides, their representations in the NTT domain are referred by ($\hat{a}$), ($\hat{\mathbf{a}}$) and ($\hat{\mathbf{A}}$), respectively. Let $\mathbf{a}$ and $\mathbf{b}$ be polynomial vectors in $\mathcal{R}_q$. We use $\mathbf{a} \circ \mathbf{b} \in \mathcal{R}_q$ to denote coefficient-wise multiplication of their polynomials. The $\circ$ product of a matrix and a vector is the natural extension of coefficient-wise multiplication of their polynomial vectors.

B. Number Theoretic Transform

The naive method of polynomial multiplication has $O(n^2)$ complexity, but it can be speeded up by using the Number Theoretic Transform. To multiply two polynomials efficiently in lattice-based cryptography, the polynomial rings of the form $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N+1)$ can be used, where (X^N+1) enables fast polynomial division. This transform maps polynomials to the NTT domain at the cost of $O(n \cdot \log n)$ where multiplying their coefficients results in a polynomial that corresponds to the product of the original polynomials modulo q and $(X^N +$

1). Coefficient-wise multiplication has a complexity of $O(n)$. Hence, a total time complexity would be $O(n \cdot \log n)$.

The NTT is a generalization of a fast Fourier transform (FFT) defined in a finite field. Suppose f is a polynomial of degree n with coefficients in $\mathbb{Z}_q$, as follows:

$$f = \sum_{i=0}^{n-1} f_i X^i \tag{1}$$

FFT uses the twiddle factor ω_n n-th root of unity of form $e^{2\pi j/n}$, while NTT has $\omega_n \in \mathbb{Z}_q$ such that ω_n be a primitive n-th root of unity modulo q, i.e. $\omega_n^n = 1 \mod q$. The NTT transforms f, i.e., $\hat{f} = NTT(f)$, is computed by Eq. 2 for each $i \in \{0, 1, ..., n-1\}$.

$$\hat{f}_i = \sum_{j=0}^{n-1} f_j \omega_n^{ij} \mod q \tag{2}$$

The INTT recovers f from $\hat{f}$ by Eq. 3 as follows:

$$f_i = \sum_{j=0}^{n-1} \hat{f}_j \omega_n^{-ij} \mod q \tag{3}$$

Hence, the multiplication between two polynomials f and g using NTT can be performed as follows:

$$f.g = INTT(NTT(f) \circ NTT(g)) \tag{4}$$

NTT algorithm is shown in Algorithm 1. Cooley-Tukey (CT) and Gentleman-Sande (GS) butterfly configurations, as illustrated in Fig. 1, can be used to facilitate NTT/INTT computation. The bit-reverse function reverses the bits of the coefficient index. However, the bit-reverse permutation can be skipped by using CT butterfly for NTT and GS for INTT [17]. Fig. 2 shows the data flow for an NTT computation of an 8-point polynomial using CT butterfly operation.

Algorithm 1 In-Place NTT Algorithm Based on Cooley-Tukey Butterfly [18]

```
Require: $a(x) \in \mathcal{R}_q, \omega_n \in \mathbb{Z}_q, n = 2^l$
Ensure: $\hat{a}(x) = NTT(a) \in \mathcal{R}_q$
1:  $\hat{a} \leftarrow$ bit-reverse($a$)
2:  for $i$ from 1 to $l$ do
3:      $m = 2^{l-i}$
4:      for $j$ from 0 to $2^{i-1} - 1$ do
5:          $W \leftarrow \omega_n^{1+j}$
6:          for $k$ from 0 to $m - 1$ do
7:              $U \leftarrow \hat{a}[2jm + k]$
8:              $V \leftarrow \hat{a}[2jm + k + m] \mod q$
9:              $T \leftarrow V \cdot W$
10:             $\hat{a}[2jm + k] = U + T \mod q$
11:             $\hat{a}[2jm + k + m] = U - T \mod q$
12:         end for
13:     end for
14: end for
15: return $\hat{a}(x) \in \mathcal{R}_q$
```

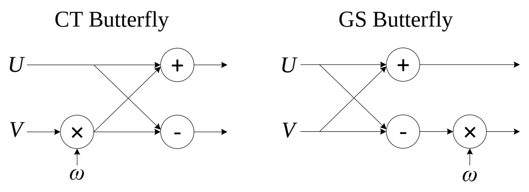

Fig. 1. Different Butterfly Configurations

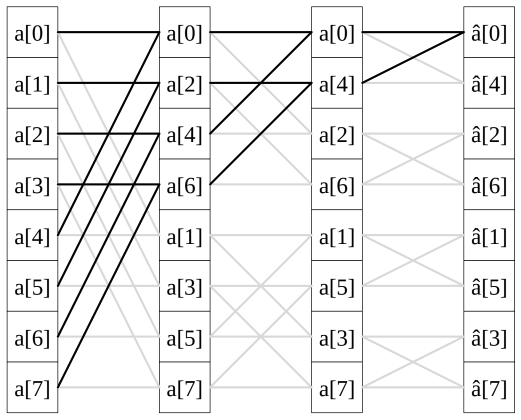

Fig. 2. Data flow of 8-point CT butterfly configuration of NTT

C. Kyber Algorithm

Kyber is an IND-CCA2-secure key encapsulation scheme [19]. It has three principal algorithms: key generation, encryption, and decryption. The algorithm samples $\mathbf{s}$ from B, and $\mathbf{A}$ from U during key generation, where B and U are binomial and uniform distributions, respectively. It computes the public key pk as $\mathbf{A} \cdot \mathbf{s} + \mathbf{e}$ in the NTT domain, where $\mathbf{e}$ is noise. In encryption, the algorithm encodes m as a polynomial and samples $\mathbf{r}$ from B. It computes $v = pk \cdot \mathbf{r} + m$ and $\mathbf{u} = \mathbf{A} \cdot \mathbf{r}$ in the normal domain. Then, it compresses $\mathbf{u}$ and v to form ciphertext ct. In decryption, the algorithm decompresses $\mathbf{u}$ and v and decodes m from $v - sk \cdot \mathbf{u}$ in the NTT domain.

All polynomials in Kyber have 256 coefficients over k-dimensional vectors and prime modulus $q = 3329$, where $k = 2, 3, 4$ denotes the three security levels, including Kyber-512 with 128-bit security, Kyber-768 with 192-bit security, and Kyber-1024 with 256-bit security. Kyber uses these polynomial functions to construct a CPA-secure PKE scheme and applies a modified Fujisaki-Okamoto transformation [20] to obtain a CCA-secure KEM.

A coefficient-wise multiplication in Kyber includes 128 modular polynomial multiplications of degree 2, such that: $(\hat{a}_{2i} + \hat{a}_{2i+1}X) \cdot (\hat{b}_{2i} + \hat{b}_{2i+1}X) = (\hat{a}_{2i}\hat{b}_{2i} + \hat{a}_{2i+1}\hat{b}_{2i+1}\omega_n^{2br_7(i)+1}) + (\hat{a}_{2i}\hat{b}_{2i+1} + \hat{a}_{2i+1}\hat{b}_{2i})X \mod (X^2 - \omega_n^{2br_7(i)+1})$, where br_7 is the bit reversal function.

III. PROPOSED ALGORITHM AND ARCHITECTURE

The most computationally intensive low-level operation in lattices is NTT. By transforming polynomials and utilizing the

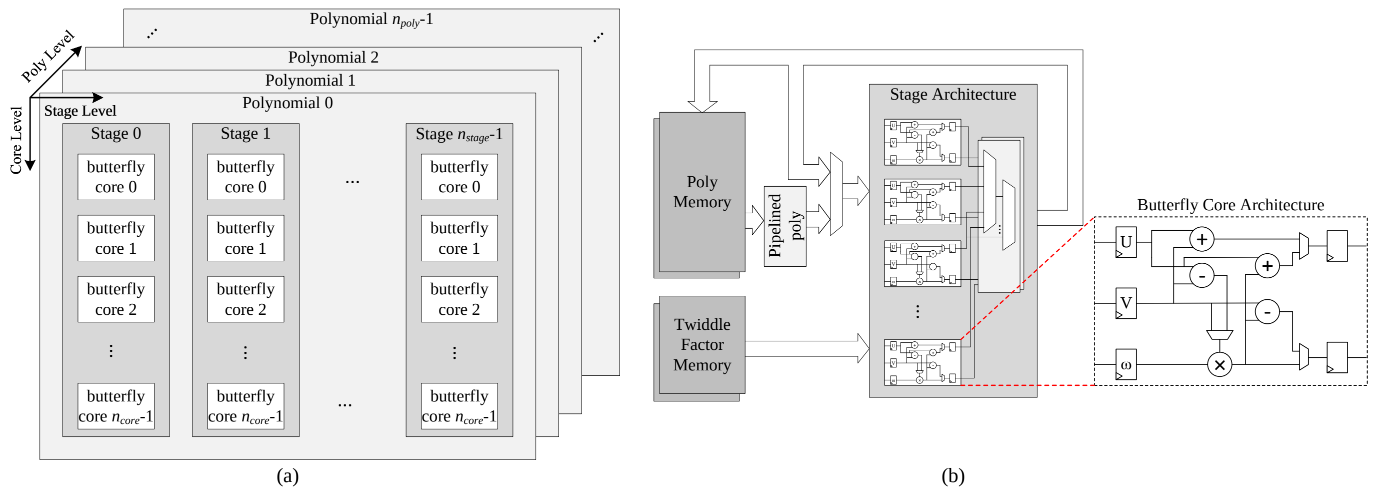

Fig. 3. Multi-level optimization of the accelerated NTT design: (a) the NTT operations in three levels, including butterfly core, stage, and polynomial levels, and (b) the proposed NTT architecture using the pipelined poly, interleaved stage architecture, and reconfigurable butterfly core.

convolution theorem, we can achieve more efficient polynomial multiplication. In this section, We first describe our proposed scalable NTT design and then our Kyber architecture. Note that we use the term NTT to refer to both NTT and INTT operations for simplicity.

A. Proposed Scalable NTT Architecture

An NTT operation can be regarded as an iterative operation by applying a sequence of butterfly operations on the input polynomial coefficients. A butterfly operation is an arithmetic operation that combines two coefficients to obtain two outputs. By repeating this process for different pairs of coefficients, the NTT operation can be computed in a logarithmic number of steps. Fig. 3 presents the high-level architecture of our proposed NTT to take advantage of FPGA-based architectural designs exploiting the multi-level of parallelism. The parallel architecture ultimately leads to improvements in the performance and efficiency of the computation. As one can see, the required operations are categorized into three levels, including butterfly core level, stage level, and polynomial level from inner to outer, respectively. We optimize each level with a different technique as follows:

1) Butterfly Core Level: A reconfigurable butterfly core is proposed to support both CT and GS operations, which are needed for NTT and INTT respectively, in order to employ resource-sharing techniques and avoid the bit-reverse cost in polynomial multiplication. To perform an NTT over a polynomial of degree n, $n/2$ independent butterfly operations per stage is required. These butterfly cores can be performed parallel to accelerate NTT operations; however, that would be challenging due to the memory access pattern, particularly, for resource-constrained platforms.

Our proposed design can be configured to set the required number of butterfly cores. This flexibility is offered by the utilization of HLS techniques. With this strategy, we involve the users in the decision-making for the trade-offs between the required resources and performance based on the target applications.

The proposed butterfly core employs three registers corresponding to each required input and also buffers the results in two output registers. Hence, the latency of the butterfly core, presented by t_{core}, is 2 cycles.

2) Stage Level: The NTT computation of a polynomial of degree n consists of $\log n$ stages, shown by n_{stage}, each of which requires the output of the preceding stage as its input. Therefore, memory access is the most crucial bottleneck in stage level implementation because the memory access pattern varies for each stage. However, NTT has an aligned access pattern, which means the number of consecutive accesses to the polynomial remains constant.

The throughput of the stage level is proportional to the number of butterfly cores. Let n_{core} be the number of implemented butterfly cores in the stage level. Given full utilization of butterfly cores, $2n_{core}$ coefficients are transformed in t_{core}. To design an interleaved stage architecture, we utilize parallel register banks embedded into the butterfly core which avoids memory access limitations during stage iterations. Since all butterfly arithmetic is modular $\mod q$, the total memory size to buffer these $2n_{core}$ coefficients in the stage architecture would be $2n_{core} \times \log q$ bits. That amount would be also equal to the throughput of this level. Simultaneously, the reordering operation needs to be performed at the stage level. We utilize an optimal multiplexer structure to pass the coefficients to the next stage. However, these multiplexers result in increasing resource consumption due to the route and placement complexity of the design.

To reduce the required hardware resources, We reuse the stage architecture to compute each stage. Hence, we feed the polynomial coefficients into butterfly cores in the first stage, and the results will be stored after t_{stage} cycles. Eq. 5 shows

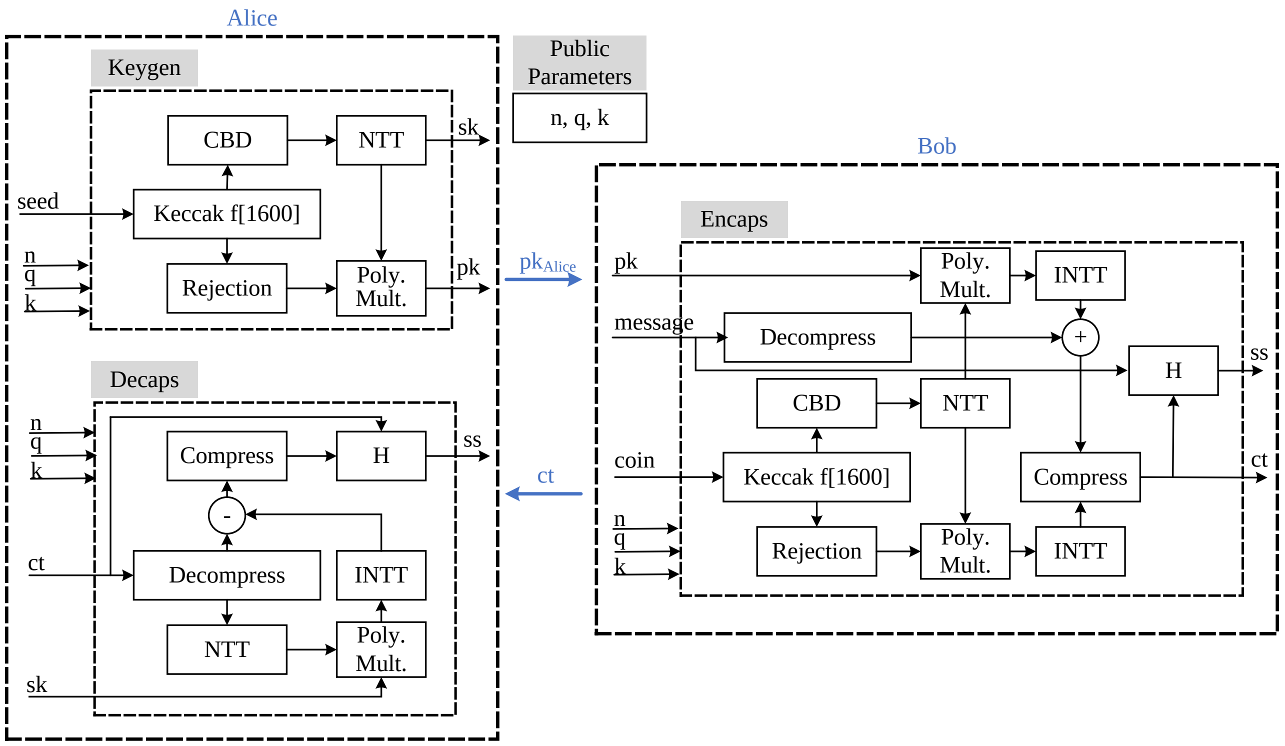

Fig. 4. High-level Kyber architecture, including our proposed scalable NTT core, coefficient-wise polynomial multiplier, Keccak-f[1600], CBD sampler, rejection sampler, and compress/decompress units (Some units are duplicated for the sake of data flow clarity, but only one set of units is implemented to support KEM operations, i.e., Keygen, Encaps, and Decaps operations.)

the required latency for each stage iteration:

$$t_{stage} = \frac{n \cdot t_{core}}{2n_{core}} \tag{5}$$

Then, the coefficients stored in register banks will be fed again into butterfly cores in the second stage, and so on. Hence, the computation of an NTT takes t_{NTT} based on Eq. 6:

$$t_{NTT} = n_{stage} \times t_{stage} = \frac{n \cdot \log n \cdot t_{core}}{2n_{core}} \tag{6}$$

3) Polynomial Level: Most of the lattice-based application requires performing NTT computation of a vector/matrix of polynomials. These operations can be performed independently. While most existing implementations use an iterative process to compute n_{poly} polynomials, we propose an optimal pipelined architecture to enhance our design from a utilization factor perspective. Using the pipelined design ensures we fully utilize stage architecture by feeding each polynomial at the last stage of the previous one. Hence, NTT computation of n_{poly} polynomials can be performed based on Eq. 7:

$$t_{poly} = n_{poly} \times t_{NTT} = \frac{n_{poly} \cdot n \cdot \log n \cdot t_{core}}{2n_{core}} \tag{7}$$

B. Proposed Kyber Architecture

The high-level data flow of Kyber architecture is presented in Fig. 4. This architecture includes NTT operation,

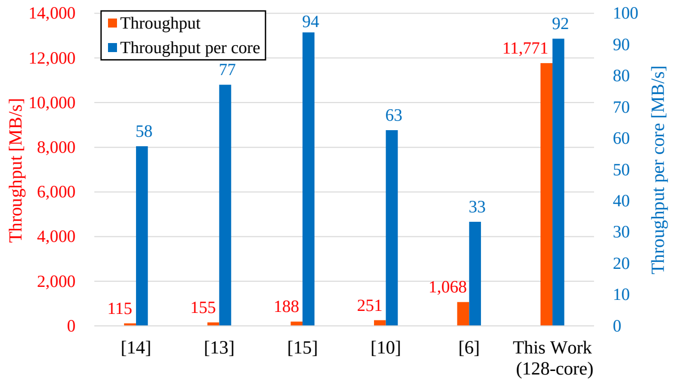

Fig. 5. Implementation results of our proposed NTT core and comparison with previous works in terms of throughput

coefficient-wise polynomial multiplier, Keccak-f[1600], binomial centered distribution (CBD), rejection sampler, and compress/decompress units. Due to the similarity of required computation in Keygen, Encaps, and Decaps operations, we implement only one set of units to support all these three operations. Using these operations, Alice and Bob can generate a shared secret key, shown by ss. Based on Kyber specification, four different configurations of Keccak are required, including SHA3-256, SHA3-512, SHAKE-128, and SHAKE-256. These functions are implemented using a configurable Keccak core providing 1600-bit output in 24 cycles in 64-bit data width.

TABLE I
IMPLEMENTATION RESULTS OF OUR PROPOSED KYBER-512 ARCHITECTURE AND COMPARISON WITH PREVIOUS WORKS

work	Platfrom	Design	Resources	Freq [MHz]	Keygen [CCs]	Encaps [CCs]	Decaps [CCs]
[12]	Virtex-7	HLS	1,977,896 LUTs/ 194,126 FFs	67	-	31,669	43,018
[14]	Artix-7	RTL	7,412 LUTs/ 4,644 FF/ 2,126 Slices/ 2 DSPs/ 3 BRAMs	161	3,768	5,079	6,668
[13]	Artix-7	RTL	18,000 LUTs/ 5,000 FFs/ 6 DSPs/ 15 BRAMs	161	4,000	7,000	10,000
[15]	Artix-7	RTL	9,347 LUTs/ 8,186 FFs/ 4 DSPs/ 6 BRAMs	220	2,100	3,300	4,500
[10]	Artix-7	RTL	10,502 LUTs/ 9,859 FFs/ 3,547 Slices/ 8 DSPs/ 13 BRAMs	200	1,882	2,446	3,754
This work (64 core)	Stratix-10	HLS	204,474 ALUTs/ 118,654 ALMs/ 78 DSPs/ 1,860 M20Ks	241	1,793	2,904	3,973

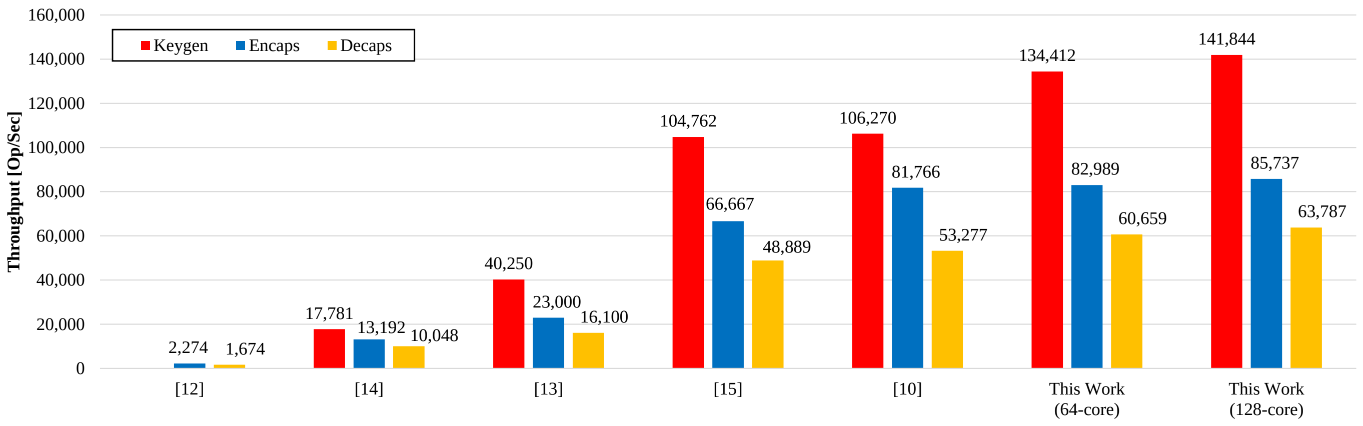

Fig. 6. Implementation results of our proposed Kyber-512 architecture and comparison with previous works in terms of performance

Our proposed NTT core is embedded into the architecture to speed up Kyber computation. Since the number of butterfly cores is configurable, the performance of Kyber can be adjusted based on the application requirement. For example, we implement two different parameter sets for the NTT core, i.e., $n_{core} = 64, 128$. Furthermore, to speed up the polynomial multiplication in the case of Kyber computation, we perform NTT separately for odd and even coefficients due to the Kyber NTT definition.

IV. IMPLEMENTATION RESULTS AND COMPARISON

Our proposed architecture is implemented on an Intel Stratix-10 FPGA device. The diversity of platforms and utilized resources used in the implementations complicate a fair and meaningful discussion or comparison of different designs and implementations with previous work. However, we attempt to put our results in the context of existing implementations to provide the reader with a quick overview of other designs and architectures.

A. Implementation Results of NTT Core

Fig. 5 depicts the implementation results of our proposed scalable NTT core for a polynomial of degree 256, i.e., $n = 256$. To have a fair comparison, we report the performance in terms of throughput in MB/s to consider different datapath widths.

We set $n_{core} = 128$ in our implementation that results in $t_{stage} = 2$ cycles based on Eq. 5. Hence, an NTT computation takes only 14 cycles, i.e., $t_{NTT} = 14$ based on Eq. 6. The results show our proposed NTT architecture achieves a throughput of 11,771 MB/s using 128 butterfly cores, while each core provides 92 MB/s on average. Our proposed HLS-based butterfly core performance results are comparable to the hand-optimized core proposed in [15], while can outperforms other architectures in [10], [13], and [14]. Compared to HLS-based NTT design in [6], ours achieves almost $3\times$ speedup per core.

Taking advantage of an optimized and scalable NTT core with multi-level parallelism, our proposed design shows a significant improvement. We achieve $11\times$ more throughput at the cost of around $4\times$ resources compared to [6]. Hence, our architecture approximately improves 63% efficiency for NTT computation. For hand-optimized RTL design, the most high-performance design is presented by Bisheh-Niasar et. al. in [10] with a merged NTT layer, while our proposed architecture outperforms that design by almost $46\times$ speedup.

B. Implementation Results of Kyber

Table I lists the detailed resource consumption and performance results for Kyber-512. We also present our results in Fig. 6 to visualize our performance results and comparison to state-of-the-art implementations in terms of the number of operation execution per second. Our proposed design employing 64 butterfly cores performs Keygen, Encaps, and Decaps operations in 7.4, 12.0, and 16.4 us, respectively. By increasing the number of cores to 128 butterfly cores, these operations take 7.0, 11.6, and 15.6 us, respectively.

TABLE II
DESIGN EFFORT AND COMPLEXITY RESULTS OF OUR PROPOSED ARCHITECTURE AND COMPARISON WITH PREVIOUS WORKS

Architecture	Design method	Man-Hour	Consideration
NTT	Manual RTL design [6]	450	Suffered of the re-design requirement for the memory structure and the control unit for changing polynomial degree and the coefficient size.
	RISC-V [6]	290	Long environment setup process to build the RISC-V tool-chains and simulation environment, and limitation of the software platform.
	HLS [6]	60	Limited capacities for exploring larger design space due to memory partitioning issue (manual memory partitioning may be required).
	Ours	80	Provided a unique degree of flexibility that can be readily adjusted for various applications for large-scale deployment of privacy-preserving computation in clouds.
Kyber	Manual RTL design [13]	410	Utilized 2 butterfly cores
	Manual RTL design [10]	550	Utilized 4 butterfly cores
	Ours	320	Scalable architecture using up to 128 butterfly cores

As the number of utilized butterfly cores increases from 64 to 128, the latency improves around $3-6\%$ at the expense of more hardware resources. This result also presents an analysis of the trade-off between resource consumption and time performance for scaling the butterfly cores. Fig. 6 indicates that scaling the butterfly cores to a higher number would reduce the NTT operation time, but increase the hardware resource usage. It also implies that this trade-off would vary depending on the application constraints, such as the time-criticality.

The work in [10] presents a high-speed architecture of Kyber using 4 butterfly cores in 2×2 arrangement using hand-optimized RTL method. Their design performs 32,258 KEMs per second, including Encaps and Decaps operations assuming Keygen is performed offline. Our proposed Kyber architecture with 128 cores executes 36,575 KEMs per second while improving the performance by 33%, 5%, and 20% for Keygen, Encaps, and Decaps operations, respectively. However, this improvement is achieved at the cost of significant resource consumption due to using the HLS method. Next, we discuss the detailed comparison with other HLS implementations, followed by a discussion about the design effort and complexity of our scalable design compared to manual RTL coding.

Although our architecture requires more resources compared to [10]. [15], and [13], we list the resource and performance results of Basu et. al. [12] as another HLS-based design to have a better comparison. As one can see, the required resources in terms of LUT and FF are reduced, while the performance is improved by a factor of $38\times$. Note that each logic in Artix-7 and Virtex-7 slice contains four 6-input LUTs and eight flip-flops. However, each ALM contains a variety of LUT-based resources that can be divided between two combinational adaptive LUTs (ALUTs), a two-bit full adder, and four registers.

C. Design Effort and Complexity

HLS needs more resources and generates larger architecture than manual RTL coding, especially for complex designs that involve memory access. However, HLS also offers some advantages such as faster development time, a higher level of abstraction, and easier verification.

Fig. 7 depicts the conceptual difference in FPGA design flow using RTL and HLS methods. Although the software

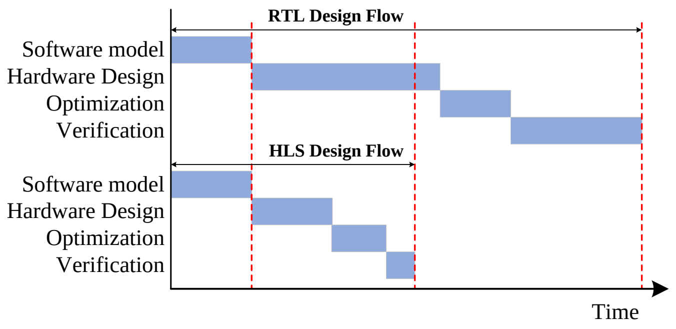

Fig. 7. FPGA design flow comparison in terms of design effort and complexity between RTL and HLS methods

model is an independent process from the implementation method, HLS can significantly reduce the required time for the hardware design, development, and verification processes. It also provides more flexibility to have a scalable architecture. For example, changing the butterfly cores in NTT operation is very challenging in manual RTL coding since it needs to re-design of the memory structure and the control unit of the NTT architecture. However, HLS provides a flexible NTT generator design that takes a few minutes to adjust the parameters and synthesize the design with desired parameter set.

We report the development time of our proposed NTT design and Kyber architecture in Table II in terms of man-hours. Mert et. al. in [6] also list the required time for three different development methods, including manual RTL design, RISC-V-based architecture, and HLS. As one can see, HLS takes less time to develop, i.e., 60-80 man-hours, to explore different optimizations and provide a parametric design framework. However, as mentioned in [6], their framework had a limited capacity to explore design space with more than 8 cores, or when the polynomial has a degree greater than 1,024.

We also report the design effort for developing an entire Kyber architecture supporting all KEM operations. The authors in [10] and [13] provided us with their development time. As one can see, the Kyber design with 2 butterfly cores takes 410 man-hours. However, by increasing the number of cores to 4, the design is more complex and takes 550 man-hours. In contrast, our scalable design takes 320 man-hours providing flexibility to users for the trade-offs between the required resources and performance.

V. Conclusion

This paper proposes an HLS approach to design a pure hardware NTT architecture over the cloud, which can offer more speed and flexibility. HLS enables us to use high-level imperative programming to design a hardware architecture that can be optimized and mapped to FPGA and ASIC platforms. Our proposed framework also allows us to utilize the scalable NTT architecture to develop a high-performance Kyber architecture targeting cloud services. To the best of our knowledge, this is the first work that implements Kyber on hardware for the cloud platform. We tackle the challenges of performance, complexity, and design time by introducing a new framework for PQC cloudization. Our framework aims to design and implement a scalable and highly parallel framework based on NTT/INTT that can speed up lattice-based PQC algorithms, particularly Kyber KEM. Our results show that our proposed framework can achieve up to $11\times$ speedup compared to existing NTT architectures while keeping high security and scalability. Our implementation is constant-time by design; however, we aim to investigate the possibility of side-channel analysis attacks in our future work, as well as optimize other units to be closer to RTL implementations from a resource usage point of view.

References

[1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pp. 124–134, 1994.

[2] The White House, "National security memorandum on promoting united states leadership in quantum computing while mitigating risks to vulnerable cryptographic systems." URL: https://www.whitehouse.gov/briefing-room/statements-releases/2022/05/04/national-security-memorandum-on-promoting-united-s, 2022.

[3] The NIST PQC Team, "PQC standardization process: Announcing four candidates to be standardized, plus fourth round candidates." URL: https://csrc.nist.gov/news/2022/pqc-candidates-to-be-standardized-and-round-4, 2022.

[4] A. Langlois and D. Stehlé, "Worst-case to average-case reductions for module lattices," *Des. Codes Cryptogr.*, vol. 75, no. 3, pp. 565–599, 2015.

[5] A. C. Mert, E. Karabulut, E. Öztürk, E. Savas, M. Becchi, and A. Aysu, "A flexible and scalable NTT hardware : Applications from homomorphically encrypted deep learning to post-quantum cryptography," in *2020 Design, Automation & Test in Europe Conference & Exhibition, DATE 2020, Grenoble, France, March 9-13, 2020*, pp. 346–351, IEEE, 2020.

[6] A. C. Mert, E. Karabulut, E. Öztürk, E. Savas, and A. Aysu, "An extensive study of flexible design methods for the number theoretic transform," *IEEE Trans. Computers*, vol. 71, no. 11, pp. 2829–2843, 2022.

[7] Y. Xing and S. Li, "An efficient implementation of the newhope key exchange on fpgas," *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 67-I, no. 3, pp. 866–878, 2020.

[8] P. Kuo, Y. Chen, Y. Hsu, C. Cheng, W. Li, and B. Yang, "High performance post-quantum key exchange on fpgas," *J. Inf. Sci. Eng.*, vol. 37, no. 5, pp. 1211–1229, 2021.

[9] D. T. Nguyen, V. B. Dang, and K. Gaj, "High-level synthesis in implementing and benchmarking number theoretic transform in lattice-based post-quantum cryptography using software/hardware codesign," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications - 16th International Symposium, ARC 2020, Toledo, Spain, April 1-3, 2020, Proceedings [postponed]*, vol. 12083, pp. 247–257, Springer, 2020.

[10] M. Bisheh-Niasar, R. Azarderakhsh, and M. M. Kermani, "High-speed ntt-based polynomial multiplication accelerator for post-quantum cryptography," in *28th IEEE Symposium on Computer Arithmetic, ARITH 2021, Lyngby, Denmark, June 14-16, 2021*, pp. 94–101, IEEE, 2021.

[11] E. Alkim, H. Evkan, N. Lahr, R. Niederhagen, and R. Petri, "ISA extensions for finite field arithmetic accelerating kyber and newhope on RISC-V," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 3, pp. 219–242, 2020.

[12] K. Basu, D. Soni, M. Nabeel, and R. Karri, "NIST post-quantum cryptography- A hardware evaluation study," *IACR Cryptol. ePrint Arch.*, p. 47, 2019.

[13] M. Bisheh-Niasar, R. Azarderakhsh, and M. M. Kermani, "Instruction-set accelerated implementation of crystals-kyber," *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 68, no. 11, pp. 4648–4659, 2021.

[14] Y. Xing and S. Li, "A compact hardware implementation of cca-secure key exchange mechanism CRYSTALS-KYBER on FPGA," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2021, no. 2, pp. 328–356, 2021.

[15] V. B. Dang, K. Mohajerani, and K. Gaj, "High-speed hardware architectures and FPGA benchmarking of crystals-kyber, ntru, and saber," *IEEE Trans. Computers*, vol. 72, no. 2, pp. 306–320, 2023.

[16] W. Guo, S. Li, and L. Kong, "An efficient implementation of KYBER," *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 69, no. 3, pp. 1562–1566, 2022.

[17] P. Longa and M. Naehrig, "Speeding up the number theoretic transform for faster ideal lattice-based cryptography," in *Cryptology and Network Security - 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings*, vol. 10052, pp. 124–139, 2016.

[18] C. Du and G. Bai, "Towards efficient polynomial multiplication for lattice-based cryptography," in *IEEE International Symposium on Circuits and Systems, ISCAS 2016, Montréal, QC, Canada, May 22-25, 2016*, pp. 1178–1181, IEEE, 2016.

[19] J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS - kyber: A cca-secure module-lattice-based KEM," in *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*, pp. 353–367, IEEE, 2018.

[20] E. Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes," in *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings* (M. J. Wiener, ed.), vol. 1666 of *Lecture Notes in Computer Science*, pp. 537–554, Springer, 1999.