ECO-CRYSTALS: Efficient Cryptography CRYSTALS on Standard RISC-V ISA

Xinyi Ji*, Jiankuo Dong*, Junhao Huang†‡, Zhijian Yuan*, Wangchen Dai§, Fu Xiao*, Jingqiang Lin¶
*School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing, China
†Department of Computer Science, BNU-HKBU United International College, Zhuhai, China
‡Department of Computer Science, Hong Kong Baptist University, Hong Kong, China
§School of Cyber Science and Technology, SUN YAT-SEN University, Hangzhou, China.
¶School of Cyber Security, University of Science and Technology of China, Hefei, China.

***Abstract*—The field of post-quantum cryptography (PQC) is continuously evolving. Many researchers are exploring efficient PQC implementation on various platforms, including x86, ARM, FPGA, GPU, etc. In this paper, we present an Efficient CryptOgraphy CRYSTALS (ECO-CRYSTALS) implementation on standard 64-bit RISC-V Instruction Set Architecture (ISA). The target schemes are two winners of the National Institute of Standards and Technology (NIST) PQC competition: CRYSTALS-Kyber and CRYSTALS-Dilithium, where the two most time-consuming operations are Keccak and polynomial multiplication. Notably, this paper is the first highly-optimized assembly software implementation to deploy Kyber and Dilithium on the 64-bit RISC-V ISA. Firstly, we propose a better scheduling strategy for Keccak, which is specifically tailored for the 64-bit dual-issue RISC-V architecture. Our 24-round Keccak permutation (Keccak-p[1600,24]) achieves a 59.18% speed-up compared to the reference implementation. Secondly, we apply two modular arithmetic (Montgomery arithmetic and Plantard arithmetic) in the polynomial multiplication of Kyber and Dilithium to get a better lazy reduction. Then, we propose a flexible dual-instruction-issue scheme of Number Theoretic Transform (NTT). As for the matrix-vector multiplication, we introduce a row-to-column processing methodology to minimize the expensive memory access operations. Compared to the reference implementation, we obtain a speedup of 53.85%∼85.57% for NTT, matrix-vector multiplication, and INTT in our ECO-CRYSTALS. Finally, our ECO-CRYSTALS implementation for key generation, encapsulation, and decapsulation in Kyber achieves 399k, 448k, and 479k cycles respectively, achieving speedups of 60.82%, 63.93%, and 65.56% compared to the NIST reference implementation. Similarly, our ECO-CRYSTALS implementation for key generation, sign, and verify in Dilithium reaches 1 364k, 3 191k, and 1 369k cycles, showcasing speedups of 54.84%, 64.98%, and 57.20%, respectively.**

***Index Terms*—PQC, Kyber, Dilithium, RISC-V, Keccak, Matrix-vector multiplication**

This work was supported in part by Major Science and Technology Demonstration Project of Jiangsu Provincial Key R & D Program under Grant No. BE2022798, in part by the National Natural Science Foundation of China under Grant No. 62302238, in part by the Natural Science Foundation of Jiangsu Province under Grant No. BK20220388, in part by the Natural Science Research Project of Colleges and Universities in Jiangsu Province under Grant No. 22KJB520004, in part by the China Postdoctoral Science Foundation under Grant No. 2022M711689, in part by the Postgraduate Research and Practice Innovation Program of Jiangsu Province under Grant No. KYCX23_1079. *Corresponding author: Fu Xiao.*

I. INTRODUCTION

The rapid advancement of quantum computers presents a significant threat to modern cryptography due to their formidable attack capabilities. Shor's algorithm [1] based on quantum computers provided an exponential speedup in cracking the discrete logarithmic and large integer factorization problems in polynomial time. Consequently, the security of public-key cryptographic algorithms, reliant on the protection of private keys, is compromised. To guarantee the secure implementation of public-key cryptography, there is an urgent demand for novel algorithms capable of withstanding both quantum and classical attacks. The post-quantum cryptography (PQC) is investigated to supersede the current public-key cryptosystems to cope with the quantum challenge. The National Institute of Standards and Technology (NIST) has launched a competition to standardize post-quantum cryptosystems. This competition is now in its fourth round, with three algorithms (CRYSTALS-DILITHIUM [2], FALCON [3], SPHINCS+ [4]) being selected to be standardized for digital signature algorithms and one algorithm (CRYSTALS-KYBER [5]) being standardized for the key encapsulation mechanism. Kyber and Dilithium are a cryptographic suite for algebraic lattices.

The performance of Kyber and Dilithium has garnered significant attention during the NIST PQC Standardization Process. Researchers have explored various optimization strategies to enhance the efficiency of computationally intensive operations in lattice-based cryptography on ARM [6] [7] [8] [9] [10] [11] [12], AVX2 [13] [14], and GPU [15] [16], etc. Sanal et al. [17] was the first optimized implementation of Kyber on 64-bit ARM processors, optimizing vectorized operations and accelerating symmetric functions through cryptography extension. Becker et al. [7] shortened the canonical implementation by eliminating explicit rotations to implement the Keccak permutation on the A-profile of the ARM architecture. Abdulrahman et al. [18] implemented NTT instances on the ARMv8.1-M+Helium and AArch64+Neon architectures. Ye et al. [19] designed novel Single-Instruction-Multiple-Data (SIMD) instructions and utilized dual-issue techniques on CV32E40P to optimize the polynomial operations in 256-bit mode.

Our Contribution. RISC-V, an open-source Instruction Set

Architecture (ISA), is gaining attention from academia and industry due to its modular design and flexibility. While the existing platforms mainly concentrate on the ARM architecture [7] [18] and the efficient SIMD RISC-V processor [19], the implementation of CRYSTALS lacks support for 64-bit RISC-V platforms. Moreover, while most research in PQC focuses on polynomial multiplication, Keccak has received comparatively less attention. Faced with these problems, we propose the first efficient PQC software implementations (Kyber1024 and Dilithium5) targeting the RV64IMB ISA on VisionFive 2 with an in-order dual-issue CPU. The contributions of the paper are summarized as follows.

- Firstly, we present a better scheduling strategy for Keccak, specifically tailored for the RV64 dual-issue architecture. In addition to leveraging the efficient manipulation instructions **rori** and **andn** in RV64IMB to speed up Keccak, we refactor Keccak instruction sequences to eliminate data hazards in a dual-issue mode, thereby maximizing Keccak throughput and improving its performance. Notably, our approach ensures no memory accesses during the Keccak round. Our Keccak implementation achieves a 59.18% speed-up compared to the reference implementation. For other 64-bit platforms lacking **rori** and **andn** instruction, we also provide an efficient generic implementation that minimizes the number of **not** operations based on the lane complementing transform technique proposed in [20]. This alternative implementation results in a 51.51% speed-up compared to the reference.
- Then, two optimized modular arithmetic (Montgomery arithmetic and Plantard arithmetic) are applied to CRYSTALS considering both instruction and memory efficiency. For polynomial arithmetic, we propose a flexible dual-instruction-issue scheme for NTT with careful register allocation and module planning. The shuffled data and interleaved instructions are designed to avoid data hazards. By introducing Montgomery arithmetic and Plantard arithmetic, we can achieve a more effective lazy reduction. After a more advanced range analysis based on Montgomery multiplication proposed in [14], there is no reduction required after NTT in CRYSTALS either using Montgomery arithmetic or Plantard arithmetic. We achieve speedups of 76.76% and 85.57% for NTT and INTT in Kyber, respectively, compared to the reference implementation. The performance improvements are 75.65% and 78.96% for NTT and INTT in Dilithium.
- Thirdly, for the matrix-vector multiplication, we present a row-to-column processing methodology to minimize expensive memory access operations by transforming the row-based approach to a column-based approach. In our matrix-vector multiplication, the number of loading sk is reduced from $k \times l \times n$ times to $l \times n$ times. We also eliminate the memory consumption for the precomputed values $sk_{2t+1}\zeta^{2br_7(t)+1}$ ($0 \leq$ t $<$ 128) and the precomputed values $sk_{2t+1}\zeta^{2br_7(t)+1}$ is multiplied at once by all related computations rather than duplicate loading them following by the redundant loading sk. In the k-th polynomial multiplication, compared to the reference implementation, we obtain speedups of 78.71% and 53.85% for matrix-vector multiplication in Kyber and Dilithium, respectively.

This paper is the first highly-optimized assembly software implementation to deploy Kyber and Dilithium on the 64-bit RISC-V ISA. By integrating optimizations across core operations (Keccak, NTT and matrix-vector multiplication), our ECO-CRYSTALS of key generation, encapsulation, and decapsulation in Kyber with Plantard arithmetic outperform the NIST reference implementation with a speedup of 60.82%$\sim$65.56%. Our ECO-CRYSTALS of key generation, sign, and verify in Dilithium with Plantard arithmetic outperform the NIST reference implementation with a speedup of 54.84%$\sim$64.98%. Due to the lack of experiments with 64-bit RISC-V ISA implementations, we also compare our ECO-CRYSTALS with the existing implementations on 32-bit or 64-bit microprocessors. Overall, our ECO-CRYSTALS outperform the implementation on 32-bit RISC-V [10] by 8.44$\times\sim$9.90$\times$ and the implementation on 32-bit ARM Cortex-M4 [11] by 2.18$\times\sim$2.98$\times$. However, compared to the implementations on the 64-bit triple-issue out-of-order architecture Cortex-A75 [17] and the state-of-the-art hardware/software implementations on 32-bit platforms CV32E40P [19], our ECO-CRYSTALS are 1.65$\times\sim$1.77$\times$ and 1.96$\times\sim$2.92$\times$ slower than the work in [17] and [19], respectively. However, we believe it is normal for our implementation to be slower than theirs because their platforms have more powerful SIMD instruction set than ours.

The remainder of the paper is structured as follows. Section 2 introduces fundamental preparatory knowledge. Section 3 describes the efficient Keccak implementation designed on the RV64 dual-issue architecture. Section 4 outlines our dedicated optimization implementation scheme of Kyber and Dilithium on RISC-V ISA. Section 5 presents the experimental results.

II. Preliminary Knowledge

In this section, the Cryptographic Suite for Algebraic Lattices (Kyber and Dilithium) is briefly described. Then, the related time-consuming operations are given. Finally, we introduce the basic knowledge of the RISC-V architecture.

A. The Cryptographic Suite for Algebraic Lattices

Two cryptographic primitives make up the cryptographic suite for algebraic lattices, Kyber and Dilithium, respectively.

1) Kyber: Kyber is an IND-CCA2-secure key-encapsulation mechanism (Kyber.CCAKEM) from an IND-CPA-secure public-key encryption scheme (Kyber.CPAPKE). It is equipped with 3 parameter sets, namely Kyber512, Kyber768, and Kyber1024, to provide different levels of security. The length of plaintext n is set to 256 and parameter k is the degree of the polynomial matrix or vector. We usually change k to scale security to different levels, where k equals 2, 3, and 4 respectively. All of the symmetric primitives

with functions are instantiated from the FIPS 202 standard. Kyber.CCAKEM consists of three parts: key generation, encapsulation, and decapsulation.

2) Dilithium: Dilithium is an EUF-CMA-secure digital signature algorithm which is based on Fiat-Shamir with Aborts [21] approach. Dilithium uses 3 parameter sets, Dilithium2, Dilithium3, and Dilithium5. The only way to scale the security of Dilithium is by changing the values of (k, l), where (k, l) equals (4,4), (6,5), and (8,7) respectively. All of the symmetric primitives with functions are instantiated from the FIPS 202 standard. All algebraic operations are assumed to be over the polynomial ring R_q. The general framework of Dilithium consists of three parts: key generation, signing, and verification.

B. Keccak

Keccak is the new standard for hash functions, having won the SHA-3 competition [20] of NIST in 2012. It is a big family consisting of four cryptographic hash functions (SHA3-224, SHA3-256, SHA3-384, and SHA3-512) and two extendable output functions (SHAKE-128 and SHAKE-256). Its main structure is the sponge structure with innovative Keccak-f permutation as the underlying function and multi-rate padding as the padding rule. The width of the permutation b and the number of iterations n_r is denoted by Keccak-p[b,n_r], where $b \in \{25, 50, 100, 200, 400, 800, 1600\}$ comprise the state and n_r is a positive integer. Input and output states of the step mappings are represented as arrays of bits with dimensions 5-by-5-by-w, where w is $b/25$. The six hash functions of Keccak can be conceptually understood as different ways or modes in which the Keccak-p[1600,24] permutation is used.

A Keccak-f round consists of five step mappings called θ, ρ, ψ, χ and ι. The pseudocode of five step mappings in a round of Keccak-f[1600] is shown in Algorithm 1. $A[x, y]$ represents a specific lane in the permutation state. $C[x]$, $D[x]$ and $B[x, y]$ are intermediate variables. The symbols $\mathbf{xor}$, $\mathbf{not}$ and $\mathbf{and}$ denote the bitwise exclusive, the bitwise complement and the bitwise AND operation, respectively. The goal of ROL(a, $offset$) is to move bit a from native position i to position $i + offset$. The constants $r[x, y]$ are specified in a table and RC is a round constant with a different value in each round.

C. Modular Arithmetic

The goal of modular arithmetic is to limit the magnitude of the coefficients. The symbol $[x]_{l'}$ in Algorithm 2 and Algorithm 3 denotes $x >> l'$, while $[x]^{l'}$ denotes $x \mod 2^{l'}$, where l' denotes machine word size $16, 32$ or 64. It is commonly applied following arithmetic operations, such as multiplication or addition. Kyber and Dilithium employ the Montgomery algorithm [22] after multiplying with a precomputed root of unity. The Barrett reduction [23] is utilized to mitigate overflow after the adding operation. To ensure the accuracy of the reduced values as representatives, the precomputed roots incorporate the Montgomery factor.

Compared to the state-of-the-art signed Montgomery arithmetic, Huang et al. [9] and Aoki et al. [12] proposed two

Algorithm 1: A Keccak-f round

Input : A state array $A[x, y]$ of 5 × 5 lanes, x,y ∈ {0, 1, 2, 3, 4}
Output: An updated state array $A[x, y]$ of 5 × 5 lanes, x,y ∈ {0, 1, 2, 3, 4}

```
// θ STEP
for x = 0 to 4 do
    C[x] = A[x,0] ⊕ A[x,1] ⊕ A[x,2] ⊕ A[x,3] ⊕ A[x,4]
for x = 0 to 4 do
    D[x] = C[x−1] ⊕ ROL(C[x+1],1)
for y = 0 to 4 do
    for x = 0 to 4 do
        A[x,y] = A[x,y] ⊕ D[x]
// ρ AND ψ AND χ STEPS
for y = 0 to 4 do
    for x = 0 to 4 do
        B[y,2x+3y] = ROL(A[x,y],r[x,y])
    for x = 0 to 4 do
        A[x,y] = B[x,y] ⊕ ((not B[x+1,y]) and B[x+2,y])
// ι STEP
A[0,0] = A[0,0] ⊕ RC
```

variants of Plantard arithmetic [24] for LBC schemes. [12] used rounding operations for which there were no corresponding integer instructions on RISC-V. [10] enlarged the input range of Plantard arithmetic which enables more effective lazy reduction strategies in NTT/INTT.

Algorithm 2: Signed Montgomery multiplication [13]

Input: Two signed integers a, b, and $ab \in$ [-$q2^{l'-1}$, $q2^{l'-1}$), $q < 2^{l'-1}$.
Output: $r \equiv ab2^{-l'} \mod q$, $r \in$ (-q,q).

1: $c = ab = c_1 2^{l'} + c_0$
2: $m = [c_0 q^{-1}]_{2^{l'}}$
3: $t_1 = [mq]^{l'}$
4: $r = c_1 - t_1$

Algorithm 3: Plantard multiplication [10]

Input: Two signed integers a, b, and $ab \in (q2^{l'} - q2^{l'+\alpha}$, $2^{2l'} - q2^{l'+\alpha})$, $q < 2^{l'-\alpha-1}$, $q' = q^{-1} \mod\pm 2^{2l'}$.
Output: $r = ab(-2^{-2l'}) \mod^{\pm} q$, $r \in$ [-$\frac{q+1}{2}$,$\frac{q}{2}$).

1: $r = [([[abq']_{2l'}]^{l'} + 2\alpha)q]^{l'}$

The primary distinction between [13] and [10] lies in the varying ranges of inputs and outputs. The input range of Plantard multiplication is at least $2^{l'+\alpha} - 2$ times bigger than Montgomery multiplication where $\alpha = 3$ for the modulus $q =$

$2^{12} - 2^{10} + 2^8 + 1$ in Kyber, α = 8 for q = $2^{23} - 2^{13} + 1$ in Dilithium, and l' is equal to 16 and 32 in Kyber and Dilithium as shown in Algorithm 2 and Algorithm 3, respectively.

D. Number Theoretic Transform

NTT is a specialized version of the Fast Fourier Transform (FFT) that operates over finite fields or rings. The sequence of elements $\mathbf{f} = [f_0, f_1, ..., f_{n-1}]$ can be transformed into another sequence by formula $\mathbf{F_i} = \sum_{j=0}^{n-1} f_j \cdot \zeta^{ij}$, where ζ is the primitive n-th roots of unity. It can be restored to the original sequence by formula $f_j = n^{-1} \sum_{i=0}^{n-1} \mathbf{F_i} \cdot (\zeta^{ij})^{-1}$, called INTT. The Cooley-Tukey algorithm (CT) [25] is a widely used algorithm for efficiently computing the Discrete Fourier Transform (DFT) and its variants NTT. The Chinese Remainder Theorem (CRT) [26] provides a way to reconstruct an integer from its residues modulo several pairwise coprime moduli. The Gentleman-Sande Algorithm (GS) [27] is used in INTT to restore the sequence. Using NTT and INTT, the product of two polynomial f, g is efficiently computed as $f \cdot g$=NTT^{-1}(NTT(f) $\circ$ NTT(g)), where $\circ$ denotes the efficient pointwise multiplication.

For Kyber, the polynomial ring is $R_q = \prod_i \mathbb{Z}_q[X]/(X^n+1)$ where $q = 13 \cdot 2^8 + 1$ and $n = 256$. The polynomial $X^{256}+1$ of R factors into 128 polynomials of degree 2 modulo q where NTT domain representation exists primitive 256-th roots of unity ζ = 17 modulo q. The cyclotomic polynomial $X^{256} + 1$ splits into $X^2 - \zeta^i$ modulo q with $i = 1, 3, 5, ..., 255$. The NTT of Kyber which has $\log n - 1$ layers is an incomplete implementation for 16-bit signed integers.

For Dilithium, the polynomial ring is $R_q = \prod_i \mathbb{Z}_q[X]/(X^n + 1)$ where $q = 2^{23} - 2^{13} + 1$ and $N = 256$. The polynomial $X^{256} + 1$ of R factors into 256 polynomials of degree 1 modulo q where NTT domain representation exists primitive 512-th roots of unity r = 1753 modulo q. The cyclotomic polynomial $X^{256} + 1$ splits into linear factors $X - r^i$ modulo q with $i = 1, 3, 5, ..., 511$. The NTT of Dilithium with $\log n$ layers is a complete implementation for 32-bit signed integers.

E. RV64IMB ISA on VisionFive 2

RV64IMB denotes a RISC-V processor architecture that embraces 64-bit functionalities (RV64). It encompasses fundamental integer operations (I), facilitates integer multiplication and division (M), and employs bit manipulation instructions (B) for fine-tuned bit-level operations. In Table I, we present a comprehensive demonstration of the fundamental integer operations in RV64IMB architecture, where sext stands for sign-extend, and S and U represent arithmetic shift right and logical shift right, respectively.

Apart from the above instructions, load and store instructions are fundamental in transferring data between memory and registers, such as **lw** (Load Word) and **ld** (Load Double Word), **sw** (Store Word) and **sd** (Store Double Word). In particular, the **mulh** instruction is part of the integer multiplication extension. This instruction performs a signed multiplication of two 64-bit integers and returns the upper 64 bits of the 128-bit result.

TABLE I
RV64IMB ISA Instruction Explanation

	Instructions	Meanings
Basic operations	**add** rd,rs1,rs2	x[rd]=x[rs1] + x[rs2]
	mul rd,rs1,rs2	x[rd]=x[rs1] $\times$ x[rs2]
	addw rd,rs1,rs2	x[rd]=sext((x[rs1] + x[rs2])[31:0])
	mulw rd,rs1,rs2	x[rd]=sext((x[rs1] $\times$ x[rs2])[31:0])
Shift operations	**sll** rd,rs1,rs2	x[rd]=x[rs1] $<<$ x[rs2]
	sra rd,rs1,rs2	x[rd]=x[rs1] $>>_s$ x[rs2]
	srl rd,rs1,rs2	x[rd]=x[rs1] $>>_u$ x[rs2]
	slliw rd,rs1,shamt	x[rd]=sext((x[rs1] $<<$ shamt)[31:0])
	sraiw rd,rs1,shamt	x[rd]=sext(x[rs1][31:0] $>>_s$ shamt)
	srliw rd,rs1,rs2	x[rd]=sext(x[rs1][31:0] $>>_u$ shamt)
Logical operations	**and** rd,rs1,rs2	x[rd]=x[rs1] & x[rs2]
	or rd,rs1,rs2	x[rd]=x[rs1] — x[rs2]
	xor rd,rs1,rs2	x[rd]=x[rs1] $\oplus$ x[rs2]
Bit manipulation operations	**andn** rd,rs1,rs2	x[rd]=x[rs1] & x̃[rs2]
	orn rd,rs1,rs2	x[rd]=x[rs1] $\vert$ x̃[rs2]
	xnor rd,rs1,rs2	x[rd]=x[rs1] $\oplus$ x̃[rs2]
Bit permutation operations	**ror** rd,rs1,rs2	x[rd]=(x[rs1] $<<$ (64-x[rs2])) $\vert$ (x[rs1] $>>$ x[rs2])
	rori rd,rs1,shamt	x[rd]=(x[rs1] $<<$ (64-shamt)) $\vert$ (x[rs1] $>>$ shamt)

Our target platform VisionFive 2 is an in-order dual-issue CPU with a non-trivial pipeline that is capable of executing up to two instructions per clock cycle. If no data hazards exist between a pair of instructions, they can be issued simultaneously in the same cycle. The execution unit is fully bypassed, allowing most instructions to have a one-cycle result latency. In particular, the load instructions and multiplication instructions generally take around 3 cycles.

III. Optimized Keccak on the RV64IMB Architecture

Bertoni et al. [20], the Keccak team members, proposed a lane complementing transform to reduce the number of **not** operations from 25 to 5 in a round of Keccak-f[1600]. They also optimized the addressing in the plane-by-plane processing to minimize the required memory for computing Keccak-f. Nevertheless, reducing the number of **not** operations is not a primary concern on the RV64IMB architecture, as it includes the **andn** instruction. In this paper, we proposed two versions of Keccak implementation. The first version is based on the open-source Keccak implementation targeting the RV64 [28] with **rori** and **andn**. The second version is also based on the Keccak implementation in [28], but we managed to reduce the **not** instructions, and we believe it could be adapted to other 64-bit platforms without **rori** and **andn** instructions.

In our experimental platform, a total of 29 registers within the 32 general-purpose registers can be programmable, excluding the hard-wired zero, the return address, and the stack point. While the stack pointer is programmable, we refrain from arbitrary changes to x2 due to program execution and memory management concerns. For the 25 fixed input and output states in Keccak-p[1600,24], we employ 25 fixed registers (64-bit wide in RV64IMB) to represent each state. Among the remaining four registers (r0, r1, r2, r3) whose values can be changed arbitrarily, they are used to store the values of intermediate variables during each round ($C0$-$C3$ in [28]).

A. Implementation of Keccak on RV64IMB architecture

Better scheduling of Keccak on RV64IMB architecture. We refactor the instruction execution order of 24 Keccak

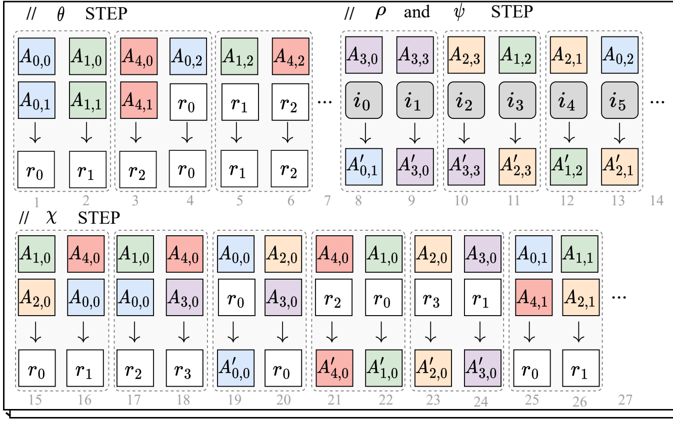

Fig. 1. Refactor Keccak instruction execution order in a dual-issue mode

rounds in a dual-issue mode. Fig. 1 shows the instruction flow in round i where indexes represent array offsets and i_0-i_5 are immediate values. Each calculation is presented as a column. Gray dotted boxes are utilized to denote the dual-issue instructions. In our platform, **xor** can be dual-issued with other **xor** instructions. The instructions **rori** and **andn** are as well. All these instructions used in Keccak can operate in dual-issue mode when there are no data conflicts. Hence, the specifics of the logical operations are omitted.

Firstly, we can fix x as a constant and keep y as a variable in the θ step of Algorithm 1. When $C[0]$ = $A[0,0] \oplus A[0,1]$ $\oplus$ ... $\oplus$ $A[0,4]$ is computed, $A[0,y]$ will be changed in the first **xor** operations with the parties of two columns where $y \in \{0,1,2,3,4\}$, as demonstrated in [28]. The computations of r0, r1 and r2 are interlaced rather than executed sequentially in the θ step, as the manual implementation is to perform line 1, line 4, and line 7 in that order.

Then, the core we are studying supports the bit manipulation extension which comes with the **rori** and **andn** instructions that are very useful when implementing Keccak. By reducing the number of instructions needed in the implementation, the **rori** and **andn** offer substantial improvements in performance, code size, and energy efficiency, particularly for Keccak requiring intensive bitwise manipulation. For the ρ and ψ step in Fig. 1, the computation result $A[0,1]$ in line 8 does not influence the source data $A[3,3]$ in line 9. It shows that there is no direct data dependency between two instructions which can be scheduled and executed in parallel.

Thirdly, for the χ step, the four available registers can store four intermediate values making multiple **andn** instructions free from resource conflicts which can maximize processor throughput and performance according to the instruction scheduling. Hence, our refactored instructions reduce the waiting time from the last computation result and avoid data hazards, ensuring the program runs in dual-issue mode. And to be sure, there are no memory accesses during the keccak round (except RC in ι step, it is inevitably), and all the interactive computation of data is performed in registers.

B. Implementation of Keccak on other 64-bit platforms

For other 64-bit platforms that do not support **rori** and **andn** instructions, the **rori** instruction can be split into one **slli**, one **srli**, and one **or** instruction, while the **andn** instruction can be split into one **not** and one **and** instruction. Notably, Bertoni et al. [20] provided the 64-bit C code implementation of the lane complementing transform without targeting the RV64IMB ISA. In particular, we apply the lane complementing transform proposed in [20] to the Keccak implementation for 64-bit RISC-V in [28]. It replaces 2 or 3 **and** operations with **or** operations and retains 5 **xor** operations where the number of **not** operations can be reduced from 600 to 120 in 24 rounds. When applying the mapping χ to the 5 lanes in a plane y, the **not** operations for each 5 lanes of the state in round i are delineated in the subsequent five equations.

$$A[1,0] = A[1,0] \ \oplus \ ((\text{not } A[2,0]) \text{ or } A[3,0]) \tag{1}$$

$$A[2,1] = A[2,1] \ \oplus \ (A[3,1] \text{ or } (\text{not } A[4,1])) \tag{2}$$

$$\begin{aligned} A[2,2] &= A[2,2] \ \oplus \ ((\text{not } A[3,2]) \text{ and } A[4,2]) \\ A[3,2] &= (\text{not } A[3,2]) \ \oplus \ (A[4,2] \text{ or } A[0,2]) \end{aligned} \tag{3}$$

$$\begin{aligned} A[2,3] &= A[2,3] \ \oplus \ ((\text{not } A[3,3]) \text{ or } A[4,3]) \\ A[3,3] &= (\text{not } A[3,3] \ \oplus \ (A[4,3] \text{ and } A[0,3]) \end{aligned} \tag{4}$$

$$\begin{aligned} A[0,4] &= A[0,4] \ \oplus \ ((\text{not } A[1,4]) \text{ and } A[2,4]) \\ A[1,4] &= (\text{not } A[1,4]) \ \oplus \ (A[2,4] \text{ or } A[3,4]) \end{aligned} \tag{5}$$

Better scheduling of Keccak on other 64-bit platforms. For one thing, the **rori** instruction is split into three instructions in the ρ and ψ step. We can interlace the **slli**, **srli**, and **or** instructions to avoid data hazards similar to the θ step in Fig. 1. For another thing, we use Equation 1 as an example to demonstrate the calculation processes across every 5 lanes in the χ step. As shown in Fig. 2, the **andn** instruction is broken down into two instructions in lines 6 and 8, separated by a single instruction to avoid stalls. Compared to the implementation of RV64IMB architecture, we use one more instruction on 64-bit platforms based on the lane complementing transform. Our process remains in a dual-issue mode to accommodate platforms that execute it out-of-order. Finally, our refactored instructions on other 64-bit platforms also eliminate memory accesses during 24 Keccak rounds.

IV. OPTIMIZED KYBER AND DILITHIUM ON THE RV64IMB ARCHITECTURE

This section discusses the detailed optimization strategies for the polynomial and matrix-vector multiplication with Montgomery and Plantard arithmetic in Kyber1024 and Dilithium5.

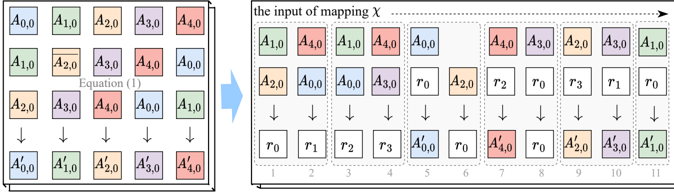

Fig. 2. Refactor Keccak instruction execution order on other 64-bit platforms

A. Montgomery and Plantard arithmetic in Kyber and Dilithium

For the choice of the Plantard algorithm, we compare two existing Plantard arithmetic in [10] and [12]. Since our target platform does not support the rounding operations required in [12], the Plantard arithmetic in [12] uses one more addition than [10]. Hence, we opt for [10]. We conduct a thorough comparison of the Montgomery arithmetic [13] and Plantard arithmetic [10], discussing from both instruction and memory perspectives. The detailed ISA is outlined in Algorithms 4, 5, 6, and 7.

1) An analysis of two arithmetic implementations in Kyber: The Montgomery arithmetic shown in [13] requires to compute a signed low product. For Kyber, this would need to fetch the low 16-bit result from a 32-bit register. Given the absence of explicit instruction for fetching the low 16 bits of 32-bit registers in the RISC-V ISA, we choose to multiply q^{-1} by 2^{16} to eliminate the high 16 bits of 32-bit registers after the **mulw** instruction rather than manually performing a 16-bit left shift followed by a 16-bit right shift. For the Plantard arithmetic, our 64-bit RISC-V platform does not support the 32-bit **mulh** instruction to multiply two 32-bit values and extract the high 32 bits of the product, and we can only implement the Plantard arithmetic in 5 instructions which is one instruction fewer than the Montgomery Multiplication. Compared to Algorithm 5, Algorithm 4 needs an extra temporary variable t to store intermediate values and an extra 32-bit register to store $q^{-1}2^{16}$.

Algorithm 4: Montgomery multiplication implementation for Kyber

Input: Two 16-bit integers a, ζ and $a\zeta \in$ [$-q2^{15}$, $q2^{15}$).
Output: $r = a\zeta 2^{-l'} \bmod q$, $r \in (-q,q)$.
1: **mulw** r, a, ζ
2: **mulw** t, r, $q^{-1}2^{16}$
3: **sraiw** t, t, 16
4: **mulw** t, t, q
5: **subw** r, r, t
6: **sraiw** r, r, 16

2) An analysis of two arithmetic implementations in Dilithium: Unlike Algorithm 4, the signed-low-product for 32-bit modulus can be directly implemented with **mulw** as shown in Algorithm 6, which saves the 64-bit product t as a 32-bit word. As for the 32-bit Plantard arithmetic

Algorithm 5: Plantard multiplication by a constant implementation for Kyber

Input: A 16-bit integer $a \in$ [-2^{15}, 2^{15}), a 32-bit precomputed twiddle factor ζ.
Output: $r = a\zeta(-2^{-2l'}) \bmod^{\pm} q$, $r \in [-\frac{q+1}{2}, \frac{q}{2})$.
1: **mulw** r, a, ζ
2: **sraiw** r, r, 16
3: **addiw** r, r, 8
4: **mulw** r, r, q
5: **sraiw** r, r, 16

in Dilithium, similar to [10], we can multiply q by q^{32} to obtain the effective high 32-bit result with **mulh** instruction. Hence, we obtain a 4-instruction Plantard multiplication by a constant (Algorithm 7) for 32-bit modulus on the 64-bit RISC-V platform which is one instruction fewer than Algorithm 6. Furthermore, compared to Algorithm 7, Algorithm 6 needs an extra temporary variable t to store intermediate values and an extra register to store q^{-1}.

Algorithm 6: Montgomery multiplication implementation for Dilithium

Input: $r = a\zeta 2^{-l'} \bmod q$, $r \in (-q,q)$.
Output: $r = a\zeta(-2^{-2l'}) \bmod^{\pm} q$, $r \in [-\frac{q+1}{2}, \frac{q}{2})$.
1: **mul** r, a, ζ
2: **mulw** t, r, q^{-1}
3: **mul** t, t, q
4: **sub** r, r, t
5: **srai** r, r, 32

Algorithm 7: Plantard multiplication by a constant implementation for Dilithium

Input: A 32-bit integer $a \in$ [-2^{31}, 2^{31}), a 64-bit precomputed twiddle factor ζ.
Output: $r = a\zeta(-2^{-2l'}) \bmod^{\pm} q$, $r \in [-\frac{q+1}{2}, \frac{q}{2})$.
1: **mul** r, a, ζ
2: **srai** r, r, 32
3: **addi** r, r, 256
4: **mulh** r, r, $q2^{32}$

3) Montgomery multiplication VS Plantard multiplication: Table II shows the number of instructions and registers used in Montgomery and Plantard multiplication on RV64 ISA in Kyber and Dilithium. Regarding the number of instructions, Plantard multiplication requires 5 instructions in Kyber and 4 instructions in Dilithium, respectively, which are all one multiplication instruction fewer than Montgomery multiplication. Regarding the number of registers, excluding the input and output parameters and the twiddle factor, Montgomery multiplication uses two more registers than Plantard multiplication in Kyber and Dilithium, in which one is fixed to store

a constant and another is used to store an intermediate value in Montgomery arithmetic.

This extra register to store the intermediate value would increase the register pressure, especially in dual-issue mode. Specifically, when the pre-allocation of registers is completed, the fixed constant in Montgomery multiplication can be reused. However, the intermediate value cannot be reused within a pair of input parameters (a_0 and a_1) that are expected to execute in dual-issue mode. To distinguish the results between $a_0\zeta$ and $a_1\zeta$, we need to use two different registers t_0 and t_1. Specifically, it would require i registers (t_0-t_i) to compute i Montgomery multiplications in dual-issue mode. In contrast, the Plantard multiplication does not need any additional registers to compute i Plantard multiplications in dual-issue mode, which significantly reduces the register pressure.

TABLE II
COMPARISON OF NUMBER OF INSTRUCTIONS AND REGISTERS BETWEEN MONTGOMERY AND PLANTARD MULTIPLICATION ON RV64 ISA

Kyber	Montgomery multiplication	Plantard multiplication
MUL (*)	3	2
ADD (+/-)	1	1
SHIFT (>>)	2	2
# of instructions	6	5
# of registers	3	1
Dilithium	**Montgomery multiplication**	**Plantard multiplication**
MUL (*)	3	2
ADD (+/-)	1	1
SHIFT (>>)	1	1
# of instructions	5	4
# of registers	3	1

B. NTT

For the dimension-256 NTT, there are 128 pairs of butterfly operations in each layer ($f = f + (g\zeta)_q$, $g = f - (g\zeta)_q$).

Register allocation. We adopt the 3+4 layer merging scheme for the 7-layer dimension-256 NTT in Kyber and the 4+4 layer merging strategy for the 8-layer dimension-256 NTT in Dilithium. The redundant registers available in RISC-V enable us to load 8 pairs of coefficients and the corresponding twiddle factors at once. For the given layers with a substantial quantity of twiddle factors, address entries of the twiddle factor array which also need to be placed in registers are utilized to differentiate the butterfly operations across layers. Our foundational principle lies in substituting as many high-frequency occurrences of variables into registers as possible, such as q and q^{-1}.

Module planning. The butterfly operation can be split into three parts: the multiplication module $g\zeta$, the reduction module $(g\zeta)_q$ (Algorithms in IV-A except the first line), and the addition and subtraction (additive) module $f = f + (g\zeta)_q$, $g = f - (g\zeta)_q$. For 8 pairs of coefficients, there are eight multiplication modules, eight reduction modules, and sixteen additive modules. It can be seen that multiplication instructions have a throughput of 1 and a 3-cycle latency [29], and multiplications cannot be dual-issued with other multiplications. It makes instruction scheduling an essential part of high-speed implementations.

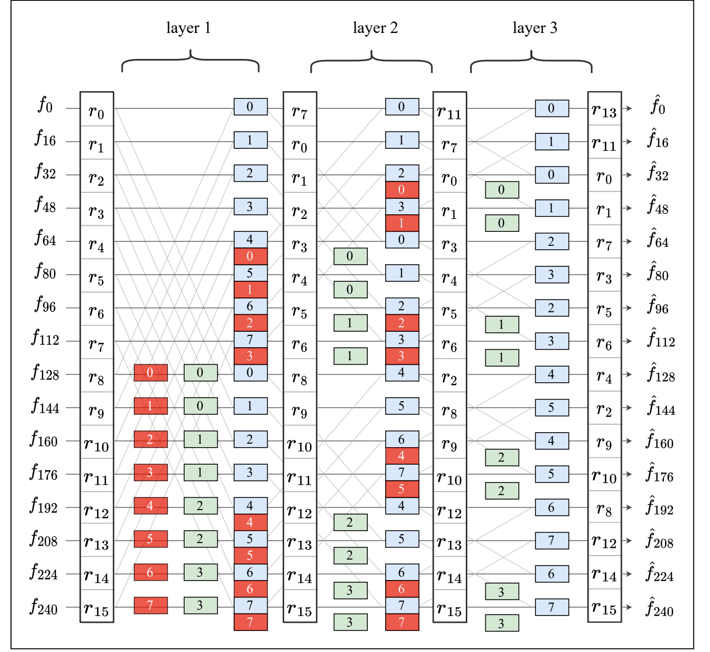

Fig. 3. A flexible dual-instruction-issue scheme of NTT

We present a flexible dual-instruction-issue scheme of NTT to avoid stalls as shown in Fig. 3 where the red, green, and blue boxes represent the multiplication module, the reduction module, and the additive module, respectively. The presence of two identical values with the same color in each column indicates that the two modules are dual-issue. The numbers represent the module operations of coefficients in each layer. Take the first three layers in NTT as an example, the group set $\{f_{i+0}, f_{i+16}, ..., f_{i+240}\}$, $0 \leq i < 16$, will be loaded into register r0-r15 in order. Once two of the multiplication modules in each layer have been completed, it is possible to advance to the reduction modules, without waiting for all the other multiplication modules in that layer to be completed.

Note that all of this is predicated on the absence of data hazards. After the reduction module, a pair of additive modules with the same number need to perform $f = f + (g\zeta)_q$, $g = f - (g\zeta)_q$. It can be seen that f will be changed in the first instruction which influences the source data of the second instruction. Hence, we shuffle the data for the 8 pairs of coefficients by using a temporary register to save a coefficient. For example, in layer 1, we can use r16 to pre-save the value of r7. Then, the register r7 can take the place of r0, thus making this formula (r7 = r0 + r8, r8 = r0 - r8) free of data conflicts. Data shuffling is performed in the additive module at each layer to ensure dual-issue capability. Ultimately, the coefficients in the registers are saved to the corresponding NTT array memory.

Moreover, a continuous sequence of multiplication instructions would be ineffective. By interleaving the order of instructions, the multiplication module of the next layer (layer 2) can be alternated with the additive module of the former

layer (layer 1) under the reasonable register allocation. When the multiplication module and the additive module appear simultaneously where there are no data hazards, they can be issued simultaneously by the processor and executed in dual-issue mode. For example in layer 1, after the fourth pair of coefficients in the additive module is calculated, the number 0 in the multiplication module can be dual-issued with the number 5 in the additive module. Suppose the twiddle factors are not pre-loaded in the registers which are ready to multiply by the coefficients of number 0 in the multiplication module, in that case, we can load them before number 4 of the additive module rather than number 0 of the multiplication module to avoid data hazards.

C. Matrix-vector Multiplication

Many optimization techniques known as asymmetric multiplication [30] and accumulation technique [8] are used to speed up pointwise multiplication and matrix-vector multiplication. The asymmetric multiplication is only applicable to Kyber. It caches the $sk_{2t+1}\zeta^{2br_7(t)+1}$ ($0 \leq$ t < 128) with a 16-bit array. The accumulation technique accumulates k pointwise multiplication, reducing the number of modular multiplications. Finally, the k-th pointwise algorithm performs modular multiplication by the sum of the products which can reduce $(k-1)\times n$ modular arithmetic.

For the matrix-vector multiplication of Kyber and Dilithium, we can express it in a unified formula:

$$pk[i][t] = a[i][j][t] \circ sk[j][t], i \in [0,k), j \in [0,l), t \in [0,n)$$

where k is 4 and 8, l is 4 and 7, in Kyber and Dilithium respectively. Note that we exclude the NTT and INTT computation in this process. The former implementations [10], [30] based on optimized techniques cache the $sk_{2t+1}\zeta^{2br_7(t)+1}$ ($0 \leq t < 128$) in memory when the first matrix-vector multiplication $pk[0][t]$ ($0 \leq t < 256$) is computed. The computation mode in the row ($pk[0][t]$, $pk[1][t]$, ..., $pk[k-1][t]$) would cause the sk to be reloaded k times despite saving the computation of $sk_{2t+1}\zeta^{2br_7(t)+1}$. Similarly, the cached memory $sk_{2t+1}\zeta^{2br_7(t)+1}$ is reloaded $k-1$ times.

We propose a row-to-column processing methodology of matrix-vector multiplication without duplicate loading and memory consumption. Fig. 4 shows the transformation from a row-based approach to a column-based approach. When the partial coefficients of $sk[j][t]$, such as $0 \leq j < l$, $0 \leq t < 2$, are loaded into the registers, we can compute the $sk[j][1]\zeta^{2br_7(0)+1}$ ($0 \leq j < l$) and execute the matrix-vector multiplication with matrix $a[0][j][t]$ ($0 \leq j < l$, $0 \leq t < 256$) to get the result of $pk[0][t]$. Note that we can not load too many coefficients at once since the limited register resources. Then, reuse the $sk[j][1]\zeta^{2br_7(0)+1}$ to continue computing with the left coefficients of $a[i][j][t]$ in matrix A which is marked with a green arrow in Fig. 4. When all related computations with $sk[j][1]\zeta^{2br_7(0)+1}$ are finished, the precomputed

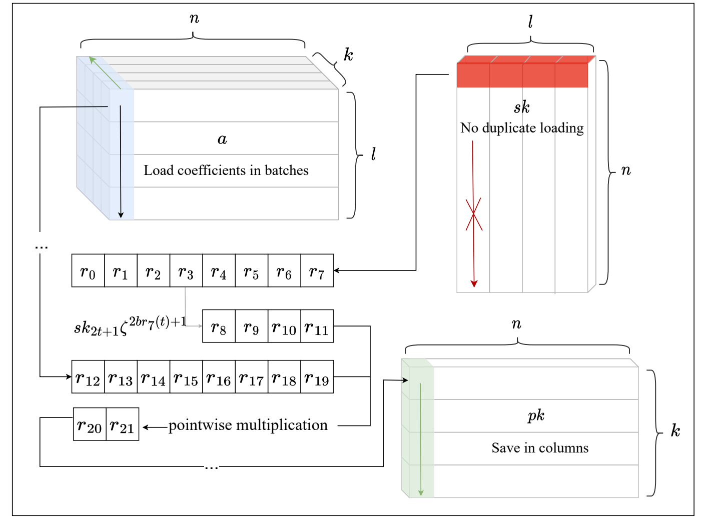

Fig. 4. A row-to-column processing methodology of matrix-vector multiplication

$sk[j][1]\zeta^{2br_7(0)+1}$ will no longer be accessed and will be released. Finally, the results are saved to pk like a column.

In our processing methodology, each coefficient of sk, a, and pk is only loaded once and the precomputed $sk[j][2t+1]\zeta^{2br_7(t)+1}$ do not consume memory, much less subsequently duplicate loading them following by duplicate loading sk. Compared with the former implementations [10], [30], We reduce the number of loading sk from $k \times l \times n$ to $l \times n$ and there is no memory consumption for $sk[j][2t+1]\zeta^{2br_7(t)+1}$. Note that there is no asymmetric multiplication in Dilithium in that we use a grey arrow to express it in Fig. 4. And, some dual-issue instructions are also possible in polynomial multiplication if the left registers are rich.

D. Lazy Reduction

In lazy reduction, instead of performing modular reduction after each basic operation, the reduction operation is postponed until it is necessary. To start, it is crucial to determine the specific contexts where reduction arithmetic is essential. Secondly, we have to specify the specific input and output ranges of two arithmetic. When employing Plantard multiplication by a constant whose value can be limited to [0,q), the input range of the coefficients can be extended to [$2^{l'}-2^{l'+\alpha}$,$2^{2l'}/q-2^{l'+\alpha}$) which is [-137q,230q] in Kyber and [-130687q,131456q] in Dilithium. Notably, the Montgomery algorithm does not support this operation because even with a fixed constant, the coefficients remain within the 16-bit range, equivalent to the parameter type range. Table III shows the maximum boundary value of the modular arithmetic.

1) NTT: We introduce a more advanced range analysis for Montgomery multiplication [14]. The output coefficients of NTT are bounded by 16540 if the inputs are less than 3329 and the output coefficients of NTT are bounded by 42082400 if the inputs are less than 8380417. Two bounds are smaller than $4.97q$ and $5.03q$ in Kyber and Dilithium respectively. For

TABLE III
THE MAXIMUM BOUNDARY VALUE OF THE MODULAR ARITHMETIC IN THE SPEED VERSION

	Output of NTT	Output of matrix-vector multiplication	Input of INTT
Kyber	$4.97q$ / $4.5q$	q / $0.5q$	q / $0.5q$
Dilithium	$5.03q$ / $5q$	q / $0.5q$	q / $0.5q$

Kyber, the coefficients after NTT are increased to $4.97q$ and $4.5q$ for Montgomery and Plantard arithmetic. We disregarded the minor increase of 0.5 in the negative range of Plantard arithmetic as it was deemed negligible. For Dilithium, one variable is sampled uniformly in [0,q) by performing rejection sampling on an array of random bytes, and the other variable is obtained from NTT. The maximum values $5.03q^2$ and $5q^2$ are within the input range of Montgomery and Plantard multiplication, respectively. Hence, neither algorithms in Kyber and Dilithium need to be reduced.

2) Matrix-vector Multiplication: Our matrix-vector multiplication is based on [6] which ensured that the output range of the matrix-vector multiplication matches that of the multiplication algorithm ($\hat{h}_{2i+1} = (\hat{f}_{2i}\hat{g}_{2i+1} + \hat{f}_{2i+1}\hat{g}_{2i})_q$). Notably, Dilithium has the polynomial of degree 1, which is unsuitable for the method in [6]. The k-th pointwise multiplication with accumulation technique [8] performs modular multiplication by the sum of the products, thereby confining the output range of the pointwise algorithm to the same range as modular arithmetic which is discussed in IV-C. This technique has been used in Kyber in [10], [30]. For Dilithium, it can be seen that $q \times 5.03q \times l$ and $q \times 5q \times l$ are within the input range of Montgomery arithmetic ($256q^2 < 2^{31}q < 257q^2$) and Plantard arithmetic ($130687q^2 < 2^{32}q - 2^{40}q < 130688q^2$) in pointwise multiplication, respectively. We can accumulate the coefficients first and delay the modular reduction to the final pointwise multiplication using either Montgomery arithmetic or Plantard arithmetic. Even though the accumulation method produces a difference of lq, it all reverts to the normal domain after the INTT operation.

3) INTT: For Kyber, the input coefficients of INTT are q with Montgomery arithmetic which will increase to $16q$ in the fourth layer. The mind of [31] is not appliable to Montgomery arithmetic where [31] computed the Fermat number transformation modulo 257 while loading the coefficients to 32-bit registers in that the input range of Montgomery arithmetic is less than $2^{15} - 1$. Not all coefficients in the fourth layer require performing modular reduction. Hence, we need to find the coefficients of $16q$ and some coefficients of $8q$ or $4q$ that may increase to $16q$ in the latter layer. Finally, the coefficients (f_{i+0}, f_{i+32}, f_{i+64}, f_{i+96}, f_{i+128}, f_{i+160}, f_{i+192}, and f_{i+224}, $0 \leq i < 16$) are required to be reduced. When implementing Kyber with Plantard arithmetic, it is sufficient that the values would not exceed the input range of Plantard multiplication by a constant ([-137q,230q]) in INTT. The coefficients are unrestricted by input ranges and parameter types. Note that the limit of parameter type 16-bit is less than the limit of the input range. Even if coefficients exceed the parameter type, they can be loaded into 64-bit registers for more efficient lazy reduction. Hence, there is no reduction required with Plantard arithmetic in Kyber. For Dilithium, there is no reduction in INTT using either Montgomery arithmetic or Plantard arithmetic.

Note that there are three reduction operations before checking the infinity norm of polynomials (chknorm function) to avoid revealing secret information in the signature of Dilithium, the coefficients require to be reduced to [-6283009,6283007]. When using the Plantard arithmetic, the maximum values are $\frac{q}{2} + 2^{19}$ which are within the range of [-6283009,6283007]. Hence, three reduction operations can be saved in the signature procedure.

V. PERFORMANCE EVALUATION

Based on our optimized RISC-V software implementation as outlined in III and IV, the experimental results for Kyber and Dilithium are presented.

A. Experimental platforms

SiFive U74 4-core. Our CPU hardware environment comprises a CPU SiFive U74, featuring a 4-core 64-bit RV64GC architecture running at 1.5GHz, with 4 GB LPDDR4 memory. The Debian GNU/Linux bookworm/sid operating system facilitates the software ecosystem, alongside GCC version 11.3.0 (Debian 11.3.0-3) as the toolchain. The power consumption is approximately 15W.
Comparative platforms. The 64-bit platforms currently available for comparison optimized for software implementation are Cortex-A55, a dual-issue out-of-order architecture and Cortex-A75, a triple-issue out-of-order architecture. Becker et al. [7] implemented Keccak while Abdulrahman et al. [18] implemented NTT. Sanal et al. [17] implemented NTT and Kyber. Due to the lack of experiments with 64-bit RISC-V ISA implementation, we also introduce the current best-performing experiment [11] based on ARMv7-M ISA implementation (Cortex-M4) and a hardware SIMD architecture [19] that supports RV32IMC ISA to compare the performance of Kyber and Dilithium. SiFive Freedom E310 is a 32-bit single-issue processor where Huang et al. [10] implemented the entire Kyber.

B. Performance of Keccak

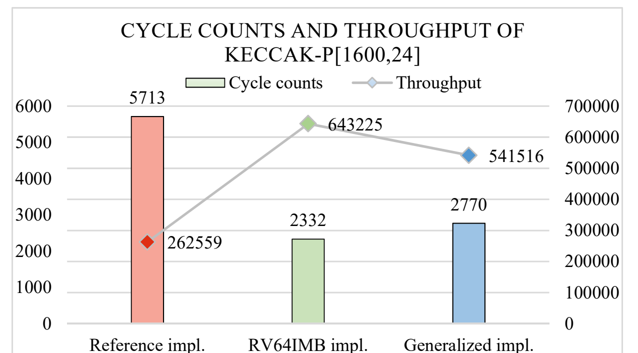

Fig. 5. Performance of Keccak-p[1600,24] on SiFive U74

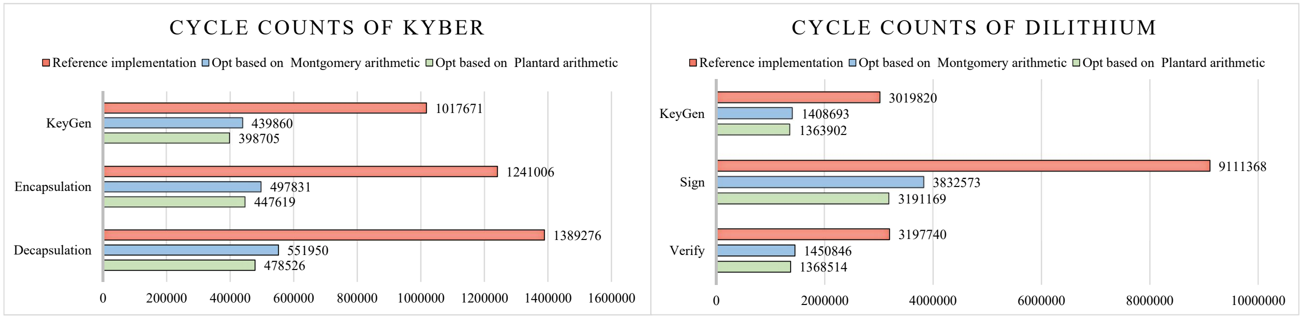

Fig. 6. Performance of Kyber and Dilithium on SiFive U74

Fig. 5 shows the cycle counts of Keccak-p[1600,24] on SiFive U74. We implement Keccak on RV64IMB ISA for Better scheduling of Keccak rounds. Except for using the **rori** and **andn** instructions, we also refactor the instruction execution order in a dual-issue path. Compared to the NIST reference implementation [32], our optimized scheduling of Keccak based on RV64IMB implementation achieves a speed-up of 59.18%. For the generalized implementation, which does not utilize the **rori** and **andn** instructions, we achieve a speed-up of 51.51%. The maximum throughput of Keccak can achieve up to 643k operations per second. However, our best performance of Keccak consumes more 914 cycle counts than [7]. The distinction between our work and [7] lies in Cortex-A55 supports substituting explicit rotations with extensive use of the Barrel shifter operation, resulting in a significant performance improvement.

C. Performance of polynomial and matrix-vector multiplication

Table IV presents the performance of NTT, matrix-vector multiplication, and INTT. For all implementations, we provide the NIST reference implementation of Kyber [5] and Dilithium [2] on 64-bit RISC-V ISA for better comparison. By introducing two arithmetic, NTT is designed in a flexible dual-instruction-issue way. Matrix-vector multiplication is transformed from a row-based approach to a column-based approach. INTT is also in a flexible dual-instruction-issue way. Compared to Montgomery arithmetic, Plantard arithmetic used 2 fewer variables which leads more flexible dual-issue.

TABLE IV
CYCLE COUNTS OF POLYNOMIAL AND MATRIX-VECTOR MULTIPLICATION

Implementation of Kyber	NTT	Matrix-vector Multiplication	INTT
Cortex-A55 (AArch64+NEON [18])	891	-	-
Cortex-A75 (AArch64+SIMD [17])	2332	-	3209
This work (C-ref [5])	24 525	214 995	38 920
This work (optimized based on Montgomery arithmetic)	8 845 63.93%	50 797 76.37%	10 262 73.63%
This work (optimized based on Plantard arithmetic)	5 700 76.76%	45 774 78.71%	5 618 85.57%
Implementation of Dilithium	**NTT**	**Matrix-vector Multiplication**	**INTT**
Cortex-A55 (AArch64+Neon [18])	1728	-	-
This work (C-ref [2])	24 202	310 898	27 258
This work (optimized based on Montgomery arithmetic)	9 131 62.27%	141 385 54.52%	9 255 66.05%
This work (optimized based on Plantard arithmetic)	5 893 75.65%	143 488 53.85%	5 736 78.96%

Multiplication instructions have a throughput of 1 and a 3-cycle latency. Because our target platform is a dual-issue platform, we can estimate a lower bound for the number of cycles for an NTT using the number of multiplications and additions. The butterfly of Kyber and Dilithium requires 2 multiplications and 5 additions with plantard multiplication which in total should require $(2 \times 3 + 5)/2 = 5.5$ cycles on the dual-issue platform. We expect the NTT with Plantard arithmetic would require 128 × 7 × 5.5 = 4928 cycles and 128 × 8 × 5.5 = 5632 cycles for the 7-layer and 8-layer NTT in Kyber and Dilithium, respectively. The current implementations take 5700 cycles and 5893 cycles which are very close to the lower bounds. Only a small fraction of the cycles would be spent on load, store, and other overhead.

Compared to the reference implementation, our proposed NTT obtains speedups of 63.93% and 76.76% in Kyber and 62.27% and 75.65% in Dilithium. Our matrix-vector multiplication achieves speedups of 76.37%/78.71% and 54.52%/53.85% for Kyber and Dilithium, respectively. In Dilithium, the matrix-vector multiplication does not need to precompute $sk_{2t+1}\zeta^{2br_7(t)+1}$, resulting in a minimal difference between using Montgomery arithmetic and Plantard arithmetic. INTT achieves speedups of 73.63%/85.57% and 66.05%/78.96% in Kyber and Dilithium, respectively. Our NTT implementations in Kyber and Dilithium are 6.40× and 3.41× slower than [18] as they reserved the Neon units for the core arithmetic. For the optimized vectorized implementation [17] of NTT/INTT in Kyber on triple-issue architecture, our implementations are 2.44×/1.75× slower than them. However, we believe it is normal for our implementation to be slower than theirs because their platforms have more powerful SIMD instruction set than ours.

D. Performance of Kyber and Dilithium

Fig. 6 shows the performance of our ECO-CRYSTALS on the 64-bit processor. We apply various optimizations focused on Keccak, polynomial and matrix-vector multiplication to the cryptographic suite. The overall reference implementation of Kyber [5] and Dilithium [2] are listed. Compared with Montgomery arithmetic, Plantard arithmetic consumes fewer numbers of instructions and performs fewer modular reductions. Hence, Plantard arithmetic demonstrates superior

performance compared to Montgomery arithmetic when employed in both Kyber and Dilithium. Finally, the cycle counts of key generation, encapsulation, and decapsulation in Kyber with Plantard arithmetic are faster by 60.82%, 63.93%, and 65.56% respectively compared to the reference implementation. And, the cycle counts of key generation, sign, and verify in Dilithium are faster by 54.84%, 64.98%, and 57.20% respectively compared to the reference implementation.

E. Comparison with Related Work

Since this paper is the first software implementation that deploys Kyber and Dilithium on a 64-bit RISC-V platform, we are unable to conduct comparisons with other implementations on the same platform. Therefore, we compare the performance of ECO-CRYSTALS with the leading performance achieved on 32-bit RISC-V [10], [19], 32-bit ARM [11] or 64-bit ARM platforms [17].

As shown in Table V, our key generation, encapsulation, and decapsulation in Kyber are 1.75×, 1.65×, and 1.77× slower than that of Cortex-A75 [17], respectively. The slow-downs mainly come from the advantage of SIMD operations on polynomial multiplication and triple-issue architecture which are not applicable to our target platform. The performance of key generation, encapsulation, and decapsulation in Kyber are 2.05×, 2.26×, and 1.96× slower than that of CV32E40P, respectively. Additionally, the performance of key generation, sign, and verify in Dilithium are 2.92×, 2.72×, and 2.55× slower than that of [19] respectively. The slow-downs are reasonable because their work is a hardware-software codesign that introduced 256-bit SIMD operations to improve the PQC performance, while our ECO-CRYSTALS is merely a software implementation, and the target platform does not support vector extension.

TABLE V
COMPARISON OF CYCLE COUNTS WITH THE STATE-OF-THE-ART WORKS

Platform	Kyber1024	KeyGen	Encapsulation	Decapsulation
Cortex-M4	Huang et al. [11]	962k	1 119k	1 043k
Cortex-A75	Sanal et al. [17]	228k	272k	271k
CV32E40P	Ye et al. [19]	195k	198k	244k
SiFive Freedom E310	Huang et al. [10]	3 794k	4 435k	4 045k
SiFive U74	This work	**399k**	**448k**	**479k**
Platform	Dilithium5	KeyGen	Sign	Verify
Cortex-M4	Huang et al. [11]	4 069k	7 730k	3 998k
CV32E40P	Ye et al. [19]	467k	1 173k	537k
SiFive U74	This work	**1 364k**	**3 191k**	**1 369k**

To the best of our knowledge, the most high-performing 32-bit RISC-V ISA implementation of Kyber currently available is provided by [10]. Compared to the implementation in [10], our key generation, encapsulation, and decapsulation in our Kyber implementation are 9.51×, 9.90×, and 8.44× faster than that of the 32-bit RISC-V ISA implementation [10], respectively. The large speed-ups mainly come from the fact that their platform is a 32-bit single-issue RISC-V platform, which are very different and incomparable from our 64-bit dual-issue platform. Compared to the best-performing implementation on the 32-bit ARM Cortex-M4 in [11], the key generation, encapsulation, and decapsulation in Kyber are 2.41×, 2.50×, and 2.18× faster than that of Cortex-M4, respectively. Additionally, the key generation, sign, and verify in Dilithium are 2.98×, 2.42×, and 2.92× faster than that of the 32-bit single-issued Cortex-M4.

VI. CONCLUSION

In this paper, we optimize the instruction execution order in a dual-issue mode for better scheduling of Keccak and employ two modular arithmetic with various optimizations to enhance polynomial multiplication. For NTT, we propose a flexible dual-instruction-issue scheme. For the matrix-vector multiplication, we introduce a row-to-column processing methodology to minimize the expensive memory access operations. Finally, the cycle counts of key generation, encapsulation, and decapsulation in Kyber are 60.82%, 63.93%, and 65.56% faster than that of NIST reference implementation, respectively. And, the cycle counts of key generation, sign, and verify in Dilithium are 54.84%, 64.98%, and 57.20% faster than that of the NIST reference implementation.

REFERENCES

[1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[2] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-dilithium: A lattice-based digital signature scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 238–268, 2018.

[3] T. Prest, P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, "Falcon," *Post-Quantum Cryptography Project of NIST*, 2020.

[4] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The sphincs+ signature framework," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 2129–2146.

[5] J. 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)*. IEEE, 2018, pp. 353–367.

[6] E. Alkim, Y. A. Bilgin, M. Cenk, and F. Gérard, "Cortex-m4 optimizations for {R, M} LWE schemes," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, pp. 336–357, 2020.

[7] H. Becker and M. J. Kannwischer, "Hybrid scalar/vector implementations of keccak and sphincs+ on aarch64," in *International Conference on Cryptology in India*. Springer, 2022, pp. 272–293.

[8] A. Abdulrahman, J.-P. Chen, Y.-J. Chen, V. Hwang, M. J. Kannwischer, and B.-Y. Yang, "Multi-moduli ntts for saber on cortex-m3 and cortex-m4," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 127–151, 2022.

[9] J. Huang, J. Zhang, H. Zhao, Z. Liu, R. C. Cheung, Ç. K. Koç, and D. Chen, "Improved plantard arithmetic for lattice-based cryptography," *Cryptology ePrint Archive*, 2022.

[10] J. Huang, H. Zhao, J. Zhang, W. Dai, L. Zhou, R. C. Cheung, Ç. K. Koç, and D. Chen, "Yet another improvement of plantard arithmetic for faster kyber on low-end 32-bit iot devices," *IEEE Transactions on Information Forensics and Security*, 2024.

[11] J. Huang, A. Adomnicăi, J. Zhang, W. Dai, Y. Liu, R. C. Cheung, Ç. K. Koç, and D. Chen, "Revisiting keccak and dilithium implementations on armv7-m," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2024, no. 2, pp. 1–24, 2024.

[12] D. Aoki, K. Minematsu, T. Okamura, and T. Takagi, "Efficient word size modular multiplication over signed integers," in *2022 IEEE 29th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2022, pp. 94–101.

[13] G. Seiler, "Faster avx2 optimized ntt multiplication for ring-lwe lattice cryptography," *Cryptology ePrint Archive*, 2018.

[14] C.-M. M. Chung, V. Hwang, M. J. Kannwischer, G. Seiler, C.-J. Shih, and B.-Y. Yang, "NTT multiplication for NTT-unfriendly rings: New speed records for Saber and NTRU on Cortex-M4 and AVX2," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 159–188, 2021.

[15] N. Gupta, A. Jati, A. K. Chauhan, and A. Chattopadhyay, "Pqc acceleration using gpus: Frodokem, newhope, and kyber," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 575–586, 2020.

[16] X. Ji, J. Dong, T. Deng, P. Zhang, J. Hua, and F. Xiao, "Hi-kyber: A novel high-performance implementation scheme of kyber based on gpu," *IEEE Transactions on Parallel and Distributed Systems*, 2024.

[17] P. Sanal, E. Karagoz, H. Seo, R. Azarderakhsh, and M. Mozaffari-Kermani, "Kyber on arm64: Compact implementations of kyber on 64-bit arm cortex-a processors," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2021, pp. 424–440.

[18] A. Abdulrahman, H. Becker, M. J. Kannwischer, and F. Klein, "Fast and clean: Auditable high-performance assembly via constraint solving," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2024, no. 1, pp. 87–132, 2024.

[19] Z. Ye, R. Song, H. Zhang, D. Chen, R. C.-C. Cheung, and K. Huang, "A highly-efficient lattice-based post-quantum cryptography processor for iot applications," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2024, no. 2, pp. 130–153, 2024.

[20] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer, "KECCAK implementation overview (May 2012)," http://keccak.noekeon.org/.

[21] V. Lyubashevsky, "Lattice signatures without trapdoors," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2012, pp. 738–755.

[22] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.

[23] P. Barrett, "Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor," in *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1986, pp. 311–323.

[24] T. Plantard, "Efficient word size modular arithmetic," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1506–1518, 2021.

[25] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[26] D. Pei, A. Salomaa, and C. Ding, *Chinese remainder theorem: applications in computing, coding, cryptography*. World Scientific, 1996.

[27] W. M. Gentleman and G. Sande, "Fast fourier transforms: for fun and profit," in *Proceedings of the November 7-10, 1966, fall joint computer conference*, 1966, pp. 563–578.

[28] Mjosaarinen, "RISC-V Cryptography Extension," https://github.com/riscv/riscv-crypto/blob/main/benchmarks/sha3/zscrypto_rv64/Keccak.c, 2024.

[29] SiFive. (2024) Vf2 software documentation. [Online]. Available: https://www.sifive.com/documentation

[30] H. Becker, V. Hwang, M. J. Kannwischer, B.-Y. Yang, and S.-Y. Yang, "Neon ntt: Faster dilithium, kyber, and saber on cortex-a72 and apple m1," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 221–244, 2022.

[31] A. Abdulrahman, V. Hwang, M. J. Kannwischer, and A. Sprenkels, "Faster Kyber and Dilithium on the Cortex-M4," in *Applied Cryptography and Network Security: 20th International Conference, ACNS 2022, Rome, Italy, June 20–23, 2022, Proceedings*. Springer, 2022, pp. 853–871.

[32] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2013, pp. 313–314.

Xinyi Ji received her B.E. degree from Nanjing University of Posts and Telecommunications in 2022, and currently pursuing a Ph.D. degree in the School of Computer Science and Technology, Nanjing University of Posts and Telecommunications. Her research interests include cryptographic engineering, lattice-based cryptography, post-quantum cryptography, and high-performance computing.

Jiankuo Dong received the B.E. degree from the Xi'an Jiaotong University, and the Ph.D. degree from the University of Chinese Academy of Sciences in 2014 and 2019, respectively. He is currently an Assistant Professor with School of Computer Science, Nanjing University of Posts and Telecommunications. His research interests include cryptographic engineering, public key cryptography and applied cryptography.

Junhao Huang received his Bachelor and Master degrees from Nanjing University of Aeronautics and Astronautics in 2018, and 2021, respectively. He is currently a PhD student at BNU-HKBU United International College and Hong Kong Baptist University. His research interests are public-key cryptography, post-quantum cryptography and cryptographic engineering.

Zhijian Yuan received his B.E. degree from Nanjing University of Information Science and Technology in 2022, and currently pursuing an MS. degree in the School of Computer Science and Technology, Nanjing University of Posts and Telecommunications. His research interests include cryptographic engineering, post-quantum cryptography, and high-performance computing.

Wangchen Dai received the Ph.D. degree in electronic engineering from the City University of Hong Kong in 2018. After that, he had appointments at Hardware Security Lab, Huawei Technologies Company Ltd., and the Department of CSSE, Shenzhen University. He is currently working as a Senior Researcher with Zhejiang Lab, Hangzhou, China. His research interests include cryptographic hardware and embedded systems, fully homomorphic encryption, and reconfigurable computing.

Fu Xiao received the Ph.D. degree in computer science and technology from the Nanjing University of Science and Technology, Nanjing, China, in 2007. He is currently a Professor and a Ph.D. Supervisor with the School of Computer Science and Technology, Nanjing University of Posts and Telecommunications. His research interest includes wireless sensor networks. Prof. Xiao is a member of the IEEE Computer Society and the Association for Computing Machinery.

Jingqiang Lin School of Cyber Security, University of Science and Technology of China, Hefei, China. Jingqiang Lin (Senior Member, IEEE) received the M.S. and Ph.D. degrees from the University of Chinese Academy of Sciences, in 2004 and 2009, respectively. He is a Full Professor with the School of Cyber Security, University of Science and Technology of China. His research interests include applied cryptography and system security.