Efficient Implementation of Kyber on Mobile Devices

Lirui Zhao*, Jipeng Zhang*, Junhao Huang[†], Zhe Liu*[‡(✉)] Gerhard Hancke[§]
*Nanjing University of Aeronautics and Astronautics, Jiangsu, China
Email: lirui.zhao@outlook.com, jp-zhang@outlook.com
jhhuang_nuaa@126.com, zhe.liu@nuaa.edu.cn, gp.hancke@cityu.edu.hk
[†]Beijing Normal University-Hong Kong Baptist University United International College, Guangdong, China
[‡]State Key Laboratory of Cryptology, P.O. Box 5159, Beijing, 100878, China
[§]City University of Hong Kong, Hong Kong, China

***Abstract*—Kyber, an IND-CCA-secure key encapsulation mechanism (KEM) based on the MLWE problem, has been shortlisted for the third round evaluation of the NIST Post-Quantum Cryptography Standardization. In this paper, we explored the optimizations of Kyber in high-performance processors from the ARM Cortex-A series, which are widely used in mainstream mobile phones. To improve the performance of Kyber, we utilized the powerful SIMD instruction set NEON in an ARMv8-A to parallelize the core modules of Kyber, i.e., modular reduction and NTT. Specifically, we specially designed the optimized implementation based on the characteristic of the NEON instruction set for the Barrett and Montgomery reduction algorithms. To make full use of the computing power of NEON instructions, we proposed a novel strategy for computing the 16-bit Barrett reduction without handling the 32-bit intermediate result. Our Barrett and Montgomery reduction showed 8.52 and 8.89 times faster than the reference implementation. As for NTT/INTT, we adopted the 2+5 layer merging strategy on an ARMv8-A to implement NTT/INTT after carefully analyzing the register occupancy of various layer merging techniques. Thanks to the selected layer merging strategy, our NTT and INTT achieved 11.89 and 13.45 times speedups compared with the reference implementation. Our optimized software achieved 1.77$\times$, 1.85$\times$, and 2.16$\times$ speedups for key generation, encapsulation, and decapsulation compared with Kyber's reference implementation.**

***Keywords*-Lattice-based Cryptography; Module-LWE; Modular Reduction; NTT; Kyber;**

I. Introduction

Quantum computing's, as one of the most advanced technologies in the modern world, supremacy has been firmly proven by recent research on quantum computers [1], [2], and officially available quantum computers are expected to be formally developed in the next few decades. Although its appearance will definitely bring many conveniences in many aspects, its development will also bring subversive changes to traditional computers and other electrical devices. Shor's algorithm [3] on a quantum computer can resolve the integer factorization problem in polynomial time, which underpins the current Public Key Cryptography (PKC), i.e., RSA, ElGamal, and Elliptic Curve Cryptography (ECC), thus raising significant concerns on the security of PKC once quantum computers have been officially developed. It is very important to develop novel types of PKC to secure massive data in the real world.

As a substitute for the traditional PKC, Post-Quantum Cryptography (PQC) is considered to be a secure PKC scheme that can resist the threat of the quantum computer. Currently, the NIST PQC standardization process is at its final round of competition. Seven finalists were selected to the final round in July 2020. Among all the seven finalists, five schemes are constructed upon the hardness of the lattice problem, i.e., Lattice-based Cryptography (LBC), which makes LBC the most promising type of scheme that would be selected as the formal PQC standard. Kyber [4], as one of the Key Encapsulation Mechanism (KEM) finalists, is one of the LBC schemes. More specifically, Kyber is constructed upon the hardness of the Module Learning-with-Errors (MLWE) problem. The module version LWE problem is a compromise problem of the LWE and Ring-LWE problem. By introducing a small l-dimensional matrix, Kyber not only has higher security than Ring-LWE-based schemes, i.e., NewHope [5], but also has better efficiency than LWE-based schemes, i.e., Classic McEliece [6]. Besides, the underlying operations, such as polynomial multiplication, Barrett reduction, and Montgomery reduction, can be reused for all parameter sets of Kyber, which gives Kyber excellent scalability. These advantages are essential factors for Kyber to become a finalist in the NIST PQC standardization.

Most of the current application scenarios use the traditional PKC as security tools. The ARM Application-profile [7] (A-profile) architecture has been widely applied to many high-performance application scenarios, such as PCs, mobile phones, gaming, and even supercomputing. When the quantum computer comes into reality, these applications will face significant challenges on their data security. Therefore, it is of great significance to deploy PQC, especially one of the finalists, Kyber, in ARM A-profile architecture to protect the data in these markets against the threat of quantum computers in the near future. ARMv8-A [8] is one of the latest 64-bit ARM A-profile architectures. As one of the most popular high-performance architectures, ARMv8-A supports both 64-bit and 128-bit NEON Single Instruc-

[✉]**Zhe Liu is the corresponding author of this paper.**

tion Multiple Data (SIMD) instruction sets [8]. The 128-bit NEON instruction set can simultaneously perform four 32-bit vector operations or eight 16-bit vector operations, which provides excellent computing power to accelerate cryptography in cryptographic engineering. However, the ARMv8-A architecture only provides 32 128-bit registers, i.e., register v0-v31. The limited register resources make it necessary to allocate registers reasonably to obtain performance improvements.

In this paper, we note that the underlying polynomial coefficients of Kyber can be stored within a 16-bit word, and most of the operations of the coefficients are irrelevant. Therefore, we aim at the ARMv8-A architecture and utilize the excellent computing power of the NEON SIMD instruction set to accelerate Kyber. Our contributions are threefold.

1) We specially designed the optimal implementation based on the characteristic of the NEON instruction set for Barrett and Montgomery reduction algorithms. We proposed a novel strategy for computing 16-bit Barrett reduction without handling the 32-bit intermediate result. Our proposed strategy dramatically improves the utilization of the computing power of the NEON instruction set, thus contributing to the 8.52 times faster Barrett reduction. As for Montgomery reduction, we made full use of the $uzp2$ instruction to combine and reduce two Montgomery results efficiently.
2) To reduce the expensive load/store instructions and fully utilize the avaliable NEON vector registers, we carefully analyzed the register occupancy of various layer merging techniques when implemented NTT with the NEON instruction set. We adopted the optimal 2+5 layer merging strategy on ARMv8-A to implement NTT. Our NTT and INTT implementations outperform Kyber's reference implementation by 11.89× and 13.45×, respectively.
3) We provided a compact and full parameter implementation for Kyber. Our optimized implementation of Kyber achieved 1.77, 1.85, and 2.16 times faster than Kyber's reference implementation on ARMv8-A.

II. Background and Related Work

A. Kyber Scheme

The security of the Kyber [4] scheme is based on the MLWE problem [9]. Different from Ring-LWE, MLWE computes on polynomial matrices and vectors. The basis of the Kyber scheme is an IND-CPA-secure public-key encryption scheme (Kyber.CPAPKE), and the IND-CCA2-secure key-encapsulation mechanism (Kyber.CCAKEM) is constructed by a slightly tweaked Fujisaki–Okamoto (FO) transformation [10]. Key generation, encryption, and decryption functions of Kyber.CPAPKE are described in Algorithm 1, 2, and 3 from a high-level perspective, and we refer readers to [4] for more details about Kyber.CCAKEM.

Algorithm 1 Kyber.CPAPKE.KeyGen [11]

```
1: $\rho, \sigma \xleftarrow{\$} \{0,1\}^{256} \times \{0,1\}^{256}$
2: $\hat{\mathbf{A}} \in \mathcal{R}_q^{k\times k} \leftarrow$ SampleUniform $(\rho)$
3: $\mathbf{s}, \mathbf{e} \in \mathcal{R}_q^k \leftarrow \mathrm{SampleCBD}(\sigma)$
4: $\hat{\mathbf{t}} \leftarrow \hat{\mathbf{A}} \circ \mathrm{NTT}(\mathbf{s}) + \mathrm{NTT}(\mathbf{e})$
5: return $pk = (\rho, \hat{\mathbf{t}}), sk = \hat{\mathbf{s}}$
```

Algorithm 2 Kyber.CPAPKE.Enc [11]

```
Require: Public key $pk = (\rho, \hat{\mathbf{t}})$
Require: Message $m \in \mathcal{R}_q$
Require: Random coins $r \in \{0,1\}^{256}$
Ensure: Ciphertext $(\mathbf{u}', v')$
1: $\hat{\mathbf{A}} \in \mathcal{R}_q^{k\times k} \leftarrow$ SampleUniform $(\rho)$
2: $\mathbf{r}, \mathbf{e}_1 \in \mathcal{R}_q^k \leftarrow$ SampleCBD $(r)$
3: $e_2 \in \mathcal{R}_q \leftarrow \mathrm{SampleCBD}(r)$
4: $\hat{\mathbf{r}} \leftarrow \mathrm{NTT}(\mathbf{r})$
5: $\mathbf{u} \leftarrow \mathrm{NTT}^{-1}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e_1}$
6: $v \leftarrow \mathrm{NTT}^{-1}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + e_2 + m$
7: return (Compress$(\mathbf{u})$, Compress$(v)$)
```

The most time-consuming computation of Kyber is polynomial multiplication in $\mathcal{R}_q = \mathbb{Z}_q(x)/(x^n + 1)$, where $q = 3329$, $n = 256$ for all variants of Kyber. Kyber utilizes the Number Theoretic Transform (NTT) to speed up its polynomial multiplication, and its parameters enable a 7-layer incomplete-NTT.

B. Number Theoretic Transform (NTT)

In order to utilize NTT to speed up polynomial multiplication, the q must be a prime number. When q satisfies $q = 1 \bmod 2n$, there exists $2n$-th primitive roots of unit, and the complete-NTT is available. For the Kyber scheme, its q only satisfies $q = 1 \bmod n$, so there exists n-th primitive roots of unit but not $2n$-th primitive roots of unit. Therefore, the 7-layer incomplete-NTT is available, and the defining polynomial $X^{256} + 1$ can be factored as follows:

$$X^{256} + 1 = \prod_{i=0}^{127}(X^2 - \zeta^{2i+1}) = \prod_{i=0}^{127}(X^2 - \zeta^{2\mathrm{br}_7(i)+1})$$

where $\zeta = 17$ is the first 256-th primitive root of unit and $\mathrm{br}_7(i)$ is the bit reversal of the unsigned 7-bit integer $i = 0, 1, ..., 127$. After the NTT transformation, the polynomial $f \in \mathcal{R}_q$ is factored into 128 polynomials of degree-1, which can be represented as

$$\left(f \bmod X^2 - \zeta^{2\mathrm{br}_7(0)+1}, \ldots, f \bmod X^2 - \zeta^{2\mathrm{br}_7(127)+1}\right)$$

$$\mathrm{NTT}(f) = \hat{f} = \hat{f}_0 + \hat{f}_1 X + \cdots + \hat{f}_{255} X^{255}$$

with

Algorithm 3 Kyber.CPAPKE.Dec [11]

```
Require: Secret key $sk = \hat{\mathbf{s}}$
Require: Compressed ciphertext $(\mathbf{u}', v')$
Ensure: Message $m \in \mathcal{R}_q$
1: $\mathbf{u} \leftarrow$ Decompress $(\mathbf{u}')$
2: $v \leftarrow$ Decompress $(v')$
3: return $v - \mathrm{NTT}^{-1}(\hat{\mathbf{s}}^T \circ \mathrm{NTT}(\mathbf{u}))$
```

Algorithm 4 Signed Montgomery reduction [14]

```
Require: $0 < q < \frac{\beta}{2}$ odd , $-\frac{\beta}{2}q \leq a = a_1\beta + a_0 < \frac{\beta}{2}q$ where $0 \leq a_0 < \beta,\ \beta = 2^{16}$
Ensure: $r' \equiv \beta^{-1}a(\mathrm{mod}\ q), -q < r' < q$
1: $m \leftarrow a_0 q^{-1} \bmod {}^{\pm}\beta$          ▷ Only low-limb needed
2: $t_1 \leftarrow \left\lfloor \frac{mq}{\beta} \right\rfloor$          ▷ Only high-limb needed
3: $r' \leftarrow a_1 - t_1$
```

Algorithm 5 Signed Barrett reduction for one word [14]

```
Require: $0 \leq q < \frac{\beta}{2}, -\frac{\beta}{2} \leq a < \frac{\beta}{2}, \beta = 2^{16}$
Ensure: $r \equiv a \pmod q$ with $0 \leq r \leq q$
1: $v \leftarrow \left\lfloor \frac{2^{\lfloor \log(q) \rfloor - 1}\beta}{q} \right\rceil$
2: $t \leftarrow \left\lfloor \frac{av}{2^{\lfloor \log(q) \rfloor - 1}\beta} \right\rceil$          ▷ Only high-limb needed
3: $t \leftarrow tq \bmod \beta$          ▷ Only low-limb needed
4: $r \leftarrow a - t$
```

Algorithm 6 Specialized reduction for the Kyber prime $q = 3329 = 2^{12} - 2^9 - 2^8 + 1$

```
Require: $-2^{15} \leq a < 2^{15}$
Ensure: $r \equiv a \pmod q, 2q < r < 3q$
1: $t \leftarrow \left\lfloor \frac{a}{2^{12}} \right\rfloor$
2: $u \leftarrow a \bmod 2^{12}$          ▷ $a = t \cdot 2^{12} + u$
3: $r \leftarrow t \cdot (2^9 + 2^8 - 1) + u$          ▷ $2^{12} = 2^9 + 2^8 - 1$
```

$$\hat{f}_{2i} = \sum_{j=0}^{127} f_{2j}\zeta^{(2\mathrm{br}_7(i)+1)j}$$

$$\hat{f}_{2i+1} = \sum_{j=0}^{127} f_{2j+1}\zeta^{(2\mathrm{br}_7(i)+1)j}$$

The coefficient-wise multiplications after NTT are performed over 128 degree-1 polynomials with modulus $(X^2 - \gamma)$, where γ is the specific power of ζ. The polynomial multiplication $f \cdot g$ can be computed using NTT by $\mathrm{NTT}^{-1}(\mathrm{NTT}(f) \odot \mathrm{NTT}(g))$, where NTT^{-1} is the inverse transformation of NTT and $\odot$ is coefficient-wise multiplication.

C. Modular Reduction

Modular reduction is the core operation when computing NTT-based polynomial multiplication. Division instruction is not recommended due to its expensive and nonconstant time characteristics. The common methods to compute modular reduction are Montgomery reduction [12], Barrett reduction [13], and specialized reduction. The specialized reduction is a tailored method for a specific modulus. Seiler *et al.* [14] proposed a modified Montgomery reduction and Barrett reduction, which are more suitable for vectorized implementation as described in Algorithms 4 and 5. The advantage of these two algorithms is that only a low-limb or high-limb product is needed during the computation. The intermediate value will not extend to a wider-length data type. Therefore, they can make fuller use of vector registers than the original version.

The specialized reduction for the Kyber prime $q = 3329$ is described in Algorithm 6, which is inspired by the specialized reduction for $q = 7681$ in [14]. This method utilizes the fact that $2^{12} \equiv 2^9 + 2^8 - 1 \pmod q$. The disadvantage is that its result may not be as precise as the Barrett reduction, which would increase the number of reductions.

D. Related Work

With the advancement of the NIST PQC standardization project, there are many works related to the optimized implementation of PQC schemes.

Botros *et al.* [11] presented an optimized implementation of round-2 Kyber on Cortex-M4, and they also gave a memory-optimized implementation on resource-constrained devices. Alkim *et al.* [15] described a rather innovative optimization for NTT on Cortex-M4. They packed two 16-bit coefficients into a 32-bit register and utilized the SIMD instructions achieved faster performance than the previous implementation. Seiler [14] presented various optimizations for NTT with AVX2 instructions, and those methods are adopted by the Kyber scheme. A major contribution of this work is that the author proposed a modified version of the Montgomery reduction that is more suitable for vectorized implementation. This work improved the polynomial multiplication of NewHope and Kyber by a factor of 4.2 and 6.3 on Skylake, respectively. Streit *et al.* [16] implemented NewHope on the ARMv8-A platform using NEON instructions. Their implementation outperforms the C reference implementation of NewHope by a factor of 8.3.

Zhang *et al.* [17] designed an accelerator on a FPGA platform for polynomial multiplication of LAC, which is a round-2 candidate of the NIST PQC project. Their accelerator can greatly improve the performance of LAC's polynomial multiplication. It is worth mentioning that they also designed a specialized reduction for LAC prime $q = 251$, and their results showed that this method is very suitable for FPGA platforms.

Bürstinghaus-Steinbach *et al.* [18] integrated the Kyber scheme into mbedTLS, which is a TLS protocol implementation for embedded devices. Their results showed that Kyber performs well in mbedTLS compared with ECC variants.

Bos *et al.* [19] provided first- and high-order masked implementations for the Kyber scheme. Their masked implementations focused on the one-bit compression operation and comparing uncompressed masked polynomials with compressed public polynomials. Their performance results on an Arm Cortex-M0+ showed that their protection against side channel attack reduced performance by 2.2 times compared to an unprotected implementation. The work in [20] provided a high-speed implementation of Kyber on a RISC-V platform. Besides, there are many other works related to the optimization of NTT on various platforms, such as [21]–[25].

III. Implementation Details

As mentioned in Section II-A–II-B, the underlying arithmetic of Kyber is based on the polynomial ring $R_q(x) = \mathbb{Z}_q(x)/\left(x^n+1\right)$, where the elements are polynomials over the normal domain or the NTT domain. Each polynomial can be represented as follows:

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

The polynomial addition and subtraction are performed component-wise, with the time complexity of $O(n)$. However, polynomial multiplication is much more complex, with the time complexity of $O(n^2)$. Therefore, NTT is used to speed up polynomial multiplication, and the time complexity of NTT is $O(nlogn)$. After polynomials f and g are transformed into the NTT domain, the complex polynomial multiplication is transformed to point-wise multiplication, thus reducing its time complexity.

$$f * g = NTT^{-1}(NTT(f) \odot NTT(g)) \tag{2}$$

where $\odot$ denotes the point-wise multiplication.

A. Barrett Reduction

The Barrett reduction approximately represents $\frac{1}{q}$ by using multiplication and shift operations instead of division:

$$\frac{1}{q} \approx \frac{v}{2^k} \rightarrow v = \lfloor \frac{2^k}{q} \rceil \tag{3}$$

where $\lfloor \rceil$ means rounding to the nearest integer. And the result of the Barrett reduction is computed by:

$$a \ mod \ q \ = a - ((a * v) >> k) \cdot q \tag{4}$$

For the Kyber scheme, its parameters involved in Equation 4 are $q = 3329$, $k = 14$, and $v = 5$. Since the input of the Barrett reduction is a 16-bit integer, the output is an integer in the range of $(-q, q)$.

Listing 1 Barrett Reduction (BarR)

```
Input: va.8h = $[a_0, a_1, \ldots, a_7]$
Input: vq.h[0] = $q$ = 3329
Input: vc.8h = 1 << 10
Input: vt1, vt2 is intermediate vector register
Output: va.8h = $[a_0, a_1 \ldots, a_7]$
1: sshr vt1.8h, va.8h, 3            ▷ $t1 = a >> 3$
2: shl vt2.8h, vt1.8h, 2            ▷ $t2 = t1 << 2 = t1 * 4$
3: add vt1.8h, vt1.8h, vt2.8h       ▷ $t1 = t1 * 5$
4: add vt1.8h, vt1.8h, vc.8h        ▷ $t1 += (1 << 10)$
5: sshr vt1.8h, vt1.8h, 11          ▷ $t1 = t1 >> 11$
6: mls va.8h, vt1.8h, vq.8h         ▷ $a -= t * q$
```

In Kyber, a and v are 16-bit integers, and the computing of $a * v$ will produce a 32-bit integer. However, expanding to double-width type will reduce the utilization of vector registers in the SIMD instructions. Therefore, we try to modify the original Barrett reduction to avoid the expansion of the data width. Inspired by the optimized implementation of NewHope on ARMv8-A in [16], we propose an improved Barrett reduction as follows:

$$a \ mod \ q \ = a - (((a >> 3) * v) >> 11) \cdot q \tag{5}$$

First of all, the integer a is shifted right by 3 bits to ensure that the multiplication result of $(a >> 3) * v$ is within 16-bit. Then, the product multiplied by v is shifted right by 11 bits to get the approximate quotient of $\lfloor a/q \rfloor$. The error caused by shifting a right ahead of time is $[(111_2 = 7_{10}) * 5] \ >> \ 11 \ = \ 0$ at most. Therefore, it doesn't influence the output range of the result of the Barrett reduction. In our implementation, the multiplication $(a >> 3) * v$ doesn't extend the data type to 32-bit, but only uses the 16-bit intermediate to make full use of the bandwidth advantage of the NEON registers, as given in Listing 1. The combination of line 3 and 4 is to obtain the rounded quotient value of a/q to ensure that the output range of the Barrett reduction is between $(-q, q)$.

In Kyber, the addition of polynomial coefficients when computing INTT (inverse of NTT) is followed by the Barrett reduction. In the reference implementation of Kyber512, $n = 256$, $q = 3329$, a 7-layer incomplete-INTT is performed. The maximum and minimum of a 16-bit signed integer are 32767 and -32768, and $9q < 32767 < 10q$. The input range of the INTT is $(-q, q)$, and after finishing the computation of one layer, the range of coefficients is doubled. After the third layer of INTT, the range becomes $(-8q, 8q)$. Therefore, the Barrett reduction is needed for avoiding overflow from a 16-bit integer. According to the above analysis, only the third, sixth, and last layers need performing the Barrett reduction, as shown in Figure 1.

Compared with the Specialized reduction in Section II-C, our experimental results show that the Specialized reduction

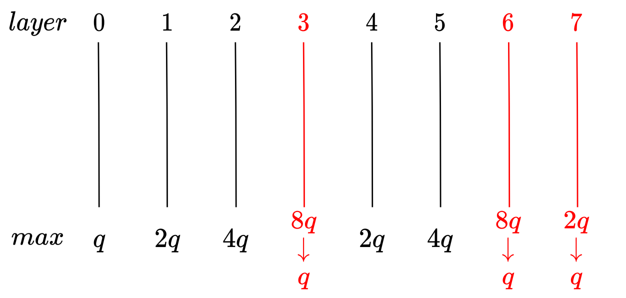


Figure 1: The position of Barrett reduction in the INTT

Listing 2 Montgomery Multiplication (MontM)

```
Input: va.8h = $[a_0, a_1, \ldots, a_7]$
Input: vb.8h = $[b_0, b_1, \ldots, b_7]$
Input: vt1, vt2 are intermediate vector registers
Output: va1.8h = $[a_0, a_1 \ldots, a_7]$
1: smull vt1.4s, va.4h, vb.4h        ▷ $t1 = (L)a * b$
2: smull2 va.4s, va.8h, vb.8h        ▷ $va = (H)a * b$
3: MontR vt1.4s, va.4s, vt2          ▷ $MontR(a * b)$
```

is faster than the Barrett reduction described in Listing 1. However, the output of the Specialized reduction range from $-2q$ to $3q$. Therefore, it is necessary to perform the Specialized reduction at the 3, 4, 5, and 6 layers, and the Barrett reduction is also needed at the last layer to ensure that the output is within $(-q, q)$. Finally, we adopt the Barrett reduction instead of the Specialized reduction, because the later will increase the number of reduction.

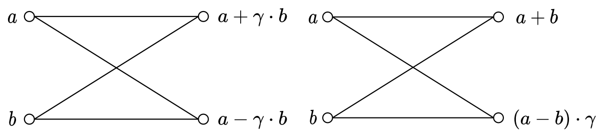
B. Montgomery Multiplication and Reduction

Montgomery multiplication is a fast modular multiplication algorithm for computing $x \cdot y \cdot \beta^{-1}\ mod\ q$, where $\beta = 2^{16}$ in our implementation.

As described in Listing 2, our Montgomery multiplication can compute eight 16-bit multiplication in parallel, i.e., $a_0 \cdot b_0, a_1 \cdot b_1, ..., a_7 \cdot b_7$, and eight 32-bit results are stored into two NEON registers. Next, Montgomery reduction (Listing 3) is used to reduce these eight 32-bit integers to eight 16-bit integers, which are finally stored into a NEON register. The right shift by 16 bits operation in line 2 of Algorithm 4, that is, taking the upper 16-bit part of the 32-bit operand, can be replaced by the $uzp2$ (Unzip vectors (secondary)) instruction in line 7 of Listing 3. This instruction reads the corresponding odd vector elements from two source registers, puts the results of the first source register into the continuous position of the lower half of the vector, puts the results of the second source register into the continuous position of the upper half of the vector, and writes the vector into the destination register. By using the $uzp2$ instruction, operations of shifting and merging into one register be implemented in one step, thus reducing the time consumption.

Listing 3 Montgomery Reduction (MontR)

```
Input: va1.4s = $[a_0, a_1, \ldots, a_3]$
Input: va2.4s = $[a_4, a_5, \ldots, a_7]$
Input: vq = q = 3329
Input: vr.4s = $[2^{16} - 1, \ldots, 2^{16} - 1]$
Input: vqp = $q^{-1}$ = 62209
Input: vt is intermediate vector register
Output: va1.8h = $[a_0, a_1 \ldots, a_7]$
1: mul vt.4s,  va1.4s, vqp          ▷ $t = a1 * q^{-1}$
2: and vt.16b, vt.16b, vr.16b       ▷ $t = (LSB)t$
3: mls va1.4s, vt.4s,  vq           ▷ $a1- = t * q$
4: mul vt.4s,  va2.4s, vqp          ▷ $t = a2 * q^{-1}$
5: and vt.16b, vt.16b, vr.16b       ▷ $t = (LSB)t$
6: mls va2.4s, vt.4s,  vq           ▷ $a2- = t * q$
7: uzp2 va1.8h, va1.8h, va2.8h      ▷ $a1 = (MSB)(a1, a2)$
```



(a) Cooley-Tukey(CT) (b) Gentleman-Sande(GS)

Figure 2: Butterfly unit

In the comments of the listings, L/H refers to the low/high four elements of a NEON register, and LSB/MSB refers to the least/most significant 16-bit of a 32-bit integer, respectively.

C. NTT Implementation

NTT and INTT are the most time-consuming modules, and their basic operation is a butterfly unit. There are two types of butterfly units, Cooley-Tukey(CT) and Gentleman-Sande(GS). The basic step of CT is shown in Figure 2a. Polynomial coefficients a and b are transformed into $a+\gamma \cdot b$ and $a - \gamma \cdot b$ by the CT butterfly unit, where γ is a specific power of ζ. The basic operation of the GS-butterfly unit is shown in Figure 2b. Polynomial coefficients a and b are transformed into $a + b$ and $(a - b) \cdot \gamma$ by the GS-butterfly unit.

The main difference between the CT and GS butterfly units lies in the order of input and output. The input of the CT butterfly unit is in normal order, and the output is in bit-reverse order, while the GS butterfly unit is the opposite. In our implementation, the NTT uses the CT butterfly unit and the INTT uses the GS butterfly unit. In this way, the bit-reverse outputs of NTT are fed to INTT, so there is no need to adjust the order of coefficients before performing INTT. As for the optimized implementation of NewHope in [16], both NTT and INTT adopt the GS butterfly unit. The advantage of their method is that the NTT and INTT can use the same code, and the disadvantage is that they need to manually adjust the order of inputs to bit-reverse the order before performing NTT and INTT.

When computing NTT, the polynomial coefficients of each layer need to be loaded and stored. There are 2^n coefficients for the n-layer NTT that will be loaded and stored $2n * 2^n$ times in total, and these memory access will consumes a lot of execution time. Therefore, the layer merging technique is used to reduce the number of $load$ and $store$ instructions to $2 * 2^n$, thereby improving the performance. Figure 3 describes a 3-layer incomplete-NTT for $n = 16$, the layer merging technique loads 16 coefficients and 8 constants to registers at one time, and then performs the 3-layer merging. In the end, these results are stored in memory.

For Kyber, $n = 256$, and there only exists 256-th primitive roots of unit. Therefore, the 7-layer incomplete-NTT is available. There are 32 128-bit registers in the NEON engine. If each register is configured as 8×16-bit, all registers can accommodate 256 16-bit values. The 7-layer incomplete-NTT of Kyber has various layer merging strategies, such as 1+6, 2+5, 3+4 layer merging. We can achieve a 6-layer merging at most, and its calculation involves 128 coefficients and 64 twiddle factors (ζ), with $16 + 8 = 24$ registers needed. Besides, the intermediate results of calculation also occupy extra registers. For example, line 1–2 in Listing 2 needs to expand the data type to store the multiplication results. When computing the 6-layer merging, Montgomery multiplications are calculated 32 times in the last layer, and 32 32-bit results will be stored into 8 128-bit registers. At this time, all 32 NEON registers are occupied, and there are no extra registers to store necessary constants (q, q^{-1}...). Hence 32 registers are not enough for the 6-layer merging. As a result, the 5-layer merging is more appropriate because we have enough registers to accommodate all 64 coefficients and 32 constants.

Therefore, we adopt a 2+5 layer merging strategy. When computing the first 2-layer merging, every 4 coefficients with a span of $256/(2^2)$ = 64 are combined as a group. In the last five layers, the 256 coefficients are divided into four blocks in sequence, and 64 coefficients of each block take the 5-layer merging.

In addition, we need to pay attention to the order of twiddle factors. The first two layers only use three twiddle factors, so the operation is relatively simple. After computing the first two layers, 256 coefficients are divided into four blocks, and we merge five layers of each block respectively. And it is necessary to adjust the order of twiddle factors to match the order of the layer merging.

IV. Performance Evaluation and Comparison

In this section, we present the performance comparison of our optimized implementation with Kyber's reference implementation. Our target platform is Raspberry Pi 3 with an ARM Cortex-A53 processor. The operating system is Ubuntu 5.4.0-1034-raspi and the GCC version is 7.5.0. Results shown in Table I and Table II are median values for 10,000 tests.

Module	ref	Our work	ref / Our work
BarR	2675	314	8.52
MontR	3413	384	8.89
NTT	16575	1394	11.89
INTT	27284	2028	13.45

Table I: Performance and Comparison of kyber512

Schemes		ref	Our work	ref / Our work
Kyber512	K	464238	262249	1.77
	E	637189	343538	1.85
	D	791471	367236	2.16
Kyber768	K	807544	484745	1.67
	E	1030702	594449	1.73
	D	1274856	641491	1.99
Kyber1024	K	1189371	783209	1.52
	E	1491847	930112	1.60
	D	1727240	1011992	1.71

Table II: Performance and Comparison of KEM

The pure C implementation of Kyber was provided in [26]. We summarized the results of the comparison in Table I and Table II. The column ref/Our work represents the speed-up of the NEON implementation over the pure C implementation.

Table I only shows the results of an important optimized module in Kyber512, because there are similar performance improvements for Kyber768, and Kyber1024. Compared with pure C implementations, the speed-ups of ref vs. Our work are reported for the modular reduction and NTT in Table I. These speed-ups are 8.52 and 8.89 for Barrett reduction and Montgomery reduction, respectively. Regarding NTT, speed-ups are 11.89 and 13.45 for NTT and INTT, respectively.

The speed-up of the overall scheme is shown in Table II. The key encapsulation mechanism (KEM) in Kyber has different reference implementations of three parameter sets. Kyber512, Kyber768, and Kyber1024 correspond to k=2, 3, and 4, respectively, and k is the dimension of the matrix of polynomials, which is the main mechanism in Kyber to scale security to different levels.

For key generation, the speed-ups vary in the range of 1.52-1.77 for Kyber. For encapsulation and decapsulation, the corresponding speed-ups vary in the range of 1.60-1.85 and 1.71-2.16 for Kyber. All speed-ups are substantially higher than for the entire implementation of Kyber.

V. Conclusion

This paper presented several optimized techniques to efficiently implement Kyber.KEM on an ARMv8. We proposed optimizations for modular reduction of Kyber and NTT operations to accelerate the execution time. First of all, the

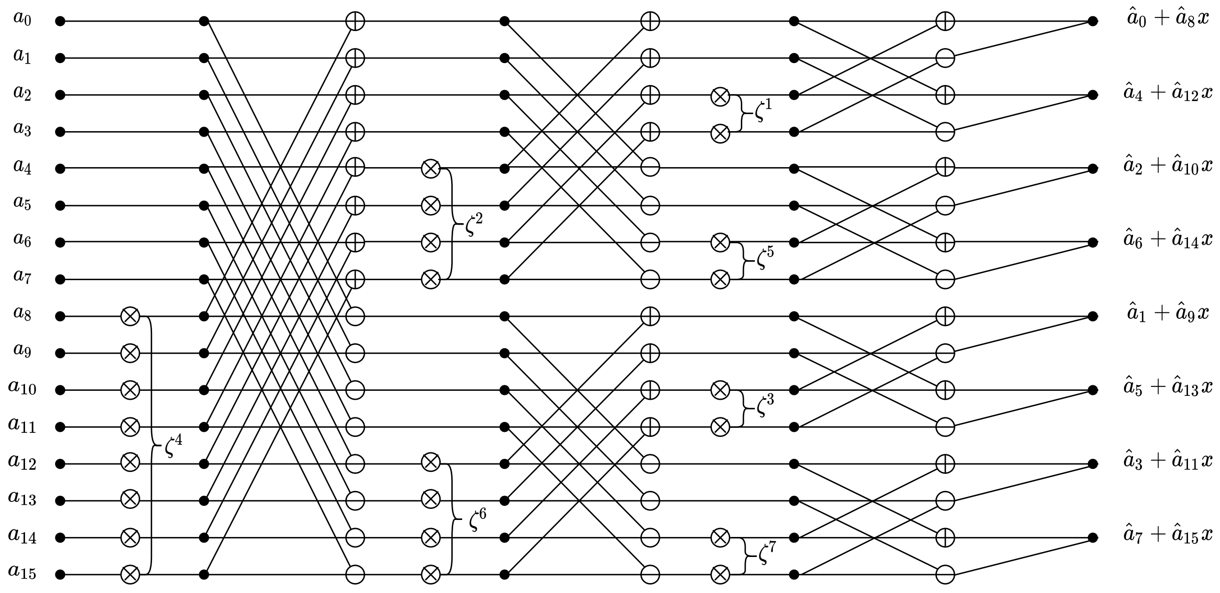

a_0
a_1
a_2
a_3
a_4
a_5
a_6
a_7
a_8
a_9
a_{10}
a_{11}
a_{12}
a_{13}
a_{14}
a_{15}
ζ^4
ζ^2
ζ^6
ζ^1
ζ^5
ζ^3
ζ^7
$\hat{a}_0 + \hat{a}_8 x$
$\hat{a}_4 + \hat{a}_{12} x$
$\hat{a}_2 + \hat{a}_{10} x$
$\hat{a}_6 + \hat{a}_{14} x$
$\hat{a}_1 + \hat{a}_9 x$
$\hat{a}_5 + \hat{a}_{13} x$
$\hat{a}_3 + \hat{a}_{11} x$
$\hat{a}_7 + \hat{a}_{15} x$

Figure 3: The NTT using Cooley-Tukey butterfly operations for n = 16

optimized Barrett reduction increases the utilization of vector registers by avoiding large data types. Secondly, NEON implementation of Montgomery multiplication and reduction greatly improves the computational efficiency of polynomial multiplication. Besides, the layer merging technique substantially accelerates the efficiency of the core module NTT. For key generation, encapsulation, and decapsulation, the combination of these optimizations achieved 1.77, 1.85, and 2.16 faster than previous Kyber512 implementations. Also, our optimizations of modular reduction and NTT operations are useful for other works, such as NewHope. These results further show the practicability of lattice-based key encapsulation mechanism to protect current ARMv8-A based platforms, like mobile phones and tablets.

Acknowledgment

Zhe Liu was supported by the National Key R&D Program of China (Grant No.2020AAA0107703), the National Key R&D Program of Guangdong Province (Grant No.2020B0101090002), the National Natural Science Foundation of China (Grant No.61802180), the Natural Science Foundation of Jiangsu Province (Grant No.BK20180421), the National Cryptography Development Fund (Grant No.MMJJ20180105), the Henan Key Laboratory of Network Cryptography Technology (Grant No.LNCT2019-A03). Jipeng Zhang was supported by 2021 Tencent Rhino-Bird Research Elite Training Program. And this work was partially supported by the Research Grants Council of Hong Kong under projects CityU 11214620.

References

[1] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[2] H.-S. Zhong, H. Wang, Y.-H. Deng, M.-C. Chen, L.-C. Peng, Y.-H. Luo, J. Qin, D. Wu, X. Ding, Y. Hu *et al.*, "Quantum computational advantage using photons," *Science*, vol. 370, no. 6523, pp. 1460–1463, 2020.

[3] 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.

[4] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Kyber (version 3.01) - Submission to round 2 of the NIST post-quantum project," *Specification document (part of the submission package)*, 2021.

[5] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Post-quantum key exchange—a new hope," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 327–343.

[6] D. J. Bernstein, T. Chou, T. Lange, I. von Maurich, R. Misoczki, R. Niederhagen, E. Persichetti, C. Peters, P. Schwabe, N. Sendrier *et al.*, "Classic McEliece: conservative code-based cryptography," *NIST submissions*, 2017.

[7] ARM Developer, "ARM Developer Documentation," https://developer.arm.com/documentation, 2021.

[8] ARM, "Introducing NEON for ARMv8-A," https://developer.arm.com/documentation/102474/0100, 2021.

[9] 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. [Online]. Available: https://doi.org/10.1007/s10623-014-9938-4

[10] 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*, ser. Lecture Notes in Computer Science, M. J. Wiener, Ed., vol. 1666. Springer, 1999, pp. 537–554. [Online]. Available: https://doi.org/10.1007/3-540-48405-1_34

[11] L. Botros, M. J. Kannwischer, and P. Schwabe, "Memory-Efficient High-Speed Implementation of Kyber on Cortex-M4," in *Progress in Cryptology - AFRICACRYPT 2019 - 11th International Conference on Cryptology in Africa, Rabat, Morocco, July 9-11, 2019, Proceedings*, ser. Lecture Notes in Computer Science, J. Buchmann, A. Nitaj, and T. Rachidi, Eds., vol. 11627. Springer, 2019, pp. 209–228. [Online]. Available: https://doi.org/10.1007/978-3-030-23696-0_11

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

[13] 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.

[14] G. Seiler, "Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography," *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 39, 2018. [Online]. Available: http://eprint.iacr.org/2018/039

[15] E. Alkim, Y. A. Bilgin, M. Cenk, and F. Gérard, "Cortex-M4 optimizations for {R, M} LWE schemes," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 3, pp. 336–357, 2020. [Online]. Available: https://doi.org/10.13154/tches.v2020.i3.336-357

[16] S. Streit and F. De Santis, "Post-quantum key exchange on ARMv8-A: A new hope for NEON made simple," *IEEE Transactions on Computers*, vol. 67, no. 11, pp. 1651–1662, 2017.

[17] J. Zhang, Z. Liu, H. Yang, J. Huang, and W. Wu, "An Efficient and Scalable Sparse Polynomial Multiplication Accelerator for LAC on FPGA," in *26th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2020, Hong Kong, December 2-4, 2020*. IEEE, 2020, pp. 390–397. [Online]. Available: https://doi.org/10.1109/ICPADS51040.2020.00059

[18] K. Bürstinghaus-Steinbach, C. Krauß, R. Niederhagen, and M. Schneider, "Post-quantum TLS on Embedded Systems: Integrating and Evaluating Kyber and SPHINCS+ with mbed TLS," in *ASIA CCS '20: The 15th ACM Asia Conference on Computer and Communications Security, Taipei, Taiwan, October 5-9, 2020*, H. Sun, S. Shieh, G. Gu, and G. Ateniese, Eds. ACM, 2020, pp. 841–852. [Online]. Available: https://doi.org/10.1145/3320269.3384725

[19] J. W. Bos, M. Gourjon, J. Renes, T. Schneider, and C. van Vredendaal, "Masking Kyber: First- and Higher-Order Implementations," *IACR Cryptol. ePrint Arch.*, vol. 2021, p. 483, 2021. [Online]. Available: https://eprint.iacr.org/2021/483

[20] D. GRECONICI, "Kyber on RISC-V," 2020.

[21] 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*, ser. Lecture Notes in Computer Science, S. Foresti and G. Persiano, Eds., vol. 10052, 2016, pp. 124–139. [Online]. Available: https://doi.org/10.1007/978-3-319-48965-0_8

[22] B. Westerbaan, "When to Barrett reduce in the inverse NTT," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 1377, 2020. [Online]. Available: https://eprint.iacr.org/2020/1377

[23] E. Karabulut and A. Aysu, "RANTT: A RISC-V Architecture Extension for the Number Theoretic Transform," in *30th International Conference on Field-Programmable Logic and Applications, FPL 2020, Gothenburg, Sweden, August 31 - September 4, 2020*, N. Mentens, L. Sousa, P. Trancoso, M. Pericàs, and I. Sourdis, Eds. IEEE, 2020, pp. 26–32. [Online]. Available: https://doi.org/10.1109/FPL50879.2020.00016

[24] C. M. Chung, V. Hwang, M. J. Kannwischer, G. Seiler, C. Shih, and B. Yang, "NTT Multiplication for NTT-unfriendly Rings New Speed Records for Saber and NTRU on Cortex-M4 and AVX2," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2021, no. 2, pp. 159–188, 2021. [Online]. Available: https://doi.org/10.46586/tches.v2021.i2.159-188

[25] M. B. Niasar, R. Azarderakhsh, and M. M. Kermani, "High-speed NTT-based Polynomial Multiplication Accelerator for CRYSTALS-Kyber Post-Quantum Cryptography," *IACR Cryptol. ePrint Arch.*, vol. 2021, p. 563, 2021. [Online]. Available: https://eprint.iacr.org/2021/563

[26] 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," https://github.com/pq-crystals/kyber.