Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.31, NO.3, Jun. 2021 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2021.31.3.463

ARM/NEON 프로세서를 활용한 NIST PQC SABER에서 Toom-Cook 알고리즘 최적화 구현 연구*

송 진 교,[1†] 김 영 범,[1] 서 석 충[2‡]
[1,2]국민대학교 금융정보보안학과 (대학원생, 교수)

Optimization Study of Toom-Cook Algorithm in NIST PQC SABER Utilizing ARM/NEON Processor*

JinGyo Song,[1†] YoungBeom Kim,[1] Seog Chung Seo[2‡]
[1,2]Department of Financial Information Security, Kookmin University
(Graduate student, Assistant Professor)

요 약

NIST(National Institute of Standards and Technology)에서는 2016년부터 양자컴퓨팅 환경을 대비하여 양자내성암호 표준화 사업을 진행하고 있다. 현재 3라운드가 진행 중이며, 대부분 후보자(5/7)는 격자기반 암호이다. 격자기반 암호는 효율적인 연산 처리와 적절한 키 길이를 제공하여 다른 기반의 양자내성 암호보다 리소스가 제한적인 임베디드 환경에서도 적용이 가능하다는 평가를 받고 있다. 그중 SABER KEM은 효율적인 모듈러스와 연산 부하가 큰 다항식 곱셈을 처리하기 위해 Toom-Cook 알고리즘을 제공한다. 본 논문에서는 ARMv8-A 환경에서 ARM/NEON을 활용하여 SABER의 Toom-Cook 알고리즘에서 평가와 보간 과정에 대한 최적화 구현 방법을 소개한다. 평가과정에서는 ARM/NEON의 효율적인 인터리빙 방법을 제안하며, 보간 과정에 서는 다양한 임베디드 환경에서 적용 가능한 최적화된 구현 방법론을 소개한다. 결과적으로 제안하는 구현은 이전 레퍼런스 구현보다 평가과정에서는 약 3.5배 보간과정에서는 약 5배 빠른 성능을 달성하였다.

ABSTRACT

Since 2016, National Institute of Standards and Technology (NIST) has been conducting a post quantum cryptography standardization project in preparation for a quantum computing environment. Three rounds are currently in progress, and most of the candidates (5/7) are lattice-based. Lattice-based post quantum cryptography is evaluated to be applicable even in an embedded environment where resources are limited by providing efficient operation processing and appropriate key length. Among them, SABER KEM provides the efficient modulus and Toom-Cook to process polynomial multiplication with computation-intensive tasks. In this paper, we present the optimized implementation of evaluation and interpolation in Toom-Cook algorithm of SABER utilizing ARM/NEON in ARMv8-A platform. In the evaluation process, we propose an efficient interleaving method of ARM/NEON, and in the interpolation process, we introduce an optimized implementation

Received(04. 21. 2021), Modified(05. 26. 2021), Accepted(05. 27. 2021)
* 본 논문은 2021년도 과학기술정보통신부의 재원으로 정보통신기획평가원의 지원(No.A2021-0270,상시적 보안품질 보장을 위한 6G 자율보안 내재화 기반기술 연구, 50%)과 2021년도 정부의 재원으로 한국연구재단의 지원(No.2019 R1F1A1068494, 50%)을 받아 수행된 연구임
† 주저자, sjk9304@kookmin.ac.kr
‡ 교신저자, scseo@kookmin.ac.kr(Corresponding)

methodology applicable in various embedded environments. As a result, the proposed implementation achieved 3.5 times faster performance in the evaluation process and 5 times faster in the interpolation process than the previous reference implementation.

Keywords: SABER, Toom-Cook, ARM/NEON, Parallel Implementation, Internet of Things

I. 서 론

2018월 3월 구글이 72 큐비트 양자컴퓨터를 개발하면서, 양자 컴퓨터의 연구 및 개발이 빠른 규모로 발전하고 있다. 하지만 이러한 양자 컴퓨팅 기술이 암호에 적용되면 Shor 알고리즘[1]을 통해 인수분해, 이산로그에 기반한 공개키 암호 시스템이 다항시간 내에 풀리게 된다. 이에 따라, NIST(National Institute of Standards and Technology)에서는 2016년 양자내성암호 공모사업[2](PKE/KEM, 전자서명)을 시작하였다. 현재 3라운드 후보자에 대해 평가과정이 진행되고 있으며, 3라운드 후보자의 대부분은 격자기반 암호로서, 계산의 효율성, 비교적 짧은 키 길이, 빠른 속도를 제공하여 High-end 컴퓨터뿐만아니라 제한적인 임베디드 환경에서도 실제 적용이 가능한 장점이 존재한다.

대표적인 격자기반 암호는 LWE(Learning With Errors)와 LWR(Learning With Rounding)에 기반하여 안전성을 보장한다. 게다가 키 길이를 효율적으로 줄이고, 256-bit 단위의 안전성을 제공하기 위해서 Module-LWE, Module-LWR 구조가 연구되었다. 실제 3라운드 후보자 중 Crystal-KYBER, Crystal-DILITHIUM은 Module-LWE 구조이며, SABER는 Module-LWR구조이다. 특히 SABER는 기존의 이항분포나 가우시안분포에서 에러를 추출하는 LWE 구조와 달리 단순히 Rounding을 적용하여 안전성을 제공한다. 게다가 SABER의 모듈러스는 2^{13}이므로 구현 시 추가적인 리덕션이 요구되지 않으며, 가장 연산 부하가 큰 다항식 곱셈에 대해 이를 빠르게 수행할 수 있는 곱셈 방법론인 Toom-Cook 알고리즘을 활용하여 빠른 처리 속도를 제공한다.

ARMv8-A series는 현재 자율주행, 핸드폰, 태블릿에 널리 사용되는 High-end MCU이다. ARMv8-A series는 64-bit ARM 프로세서와 병렬처리가 가능한 NEON 엔진을 지원한다. ARM 프로세서는 NEON 엔진과 달리 병렬처리를 지원하지 않지만, 시프트 연산에 대한 사이클을 숨길 수 있는 배럴 시프터를 지원한다. NEON 엔진은 128-bit 벡터 레지스터 내에서 8-bit, 16-bit, 32-bit, 64-bit 단위로 병렬처리가 가능하다. 따라서 최적화 구현을 통해 연산 집약적인 암호알고리즘의 성능 향상에 매우 효율적이다. 게다가 ARM과 NEON은 독립적인 모듈로 존재하여, 서로 연산을 각자 수행한다. 이를 통해 ARM과 NEON의 각 구현의 인터리빙을 통해 ARM 연산들의 지연시간을 NEON 연산의 부하로 숨길 수 있다. 위와 같은 장점들로 ARM/NEON 프로세서를 활용한 다양한 암호 최적화 연구([3],[4],[5])가 존재하지만, 아직 양자내성암호에 대해서는 미진하다. 게다가 대부분의 양자내성암호의 연구결과는 ARM Cortex-M4 환경에서만 수행하였다.

본 논문에서는 처음으로 ARMv8-A Series에서 ARM/NEON 프로세서를 활용하여 SABER의 핵심 연산인 다항식 곱셈 Toom-Cook 알고리즘의 평가와 보간 단계의 최적화 구현 방안을 제시한다. 제안하는 방법을 통해 레퍼런스 구현보다 평가과정에서는 약 3.5배, 보간과정에서는 약 5배 성능향상을 달성하였다. 본 논문의 구성은 다음과 같다. 2장에서는 Toom-Cook 알고리즘과 ARM/NEON 프로세서의 자세한 설명을 제공한다. 3장에서는 기존 ARM/NEON 프로세서의 관련 연구들을 살펴보고, 4장에서는 Toom-Cook 알고리즘의 평가 및 보간 단계의 최적화 방법을 제안한다. 5장에서는 성능 측정 결과를 제시하고 마지막으로 결론과 향후 계획으로 본 논문을 마무리한다.

II. Background

2.1 Toom-Cook 알고리즘

SABER에서는 가장 핵심 연산인 256차 다항식 곱셈 연산을 Toom-Cook 알고리즘을 통해 수행한다. Toom-Cook 알고리즘은 n차 다항식 곱셈 시 복잡도인 $O(n^2)$에서 이를 $O(n^{\log 7/\log 4})$의 복잡도를 낮춰 단순 곱셈보다 더욱 빠른 속도로 곱셈 연산을 처리한다. SABER에서는 Toom-Cook 4-way

를 사용하여 256차 다항식을 4개로 분할하며, 나머지 Took-Cook 알고리즘의 과정은 평가, 점 별 곱셈, 보간 과정을 통해 다항식 곱셈을 수행한다.

2.1.1 평가

평가과정에서는 연산이 비교적 간단한 각 포인트에 대하여 결과값을 구한다. SABER에서는 Toom-Cook-4way를 사용하였으므로, 하나의 256차 다항식 A에 대해 $A(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ 로 표현될 수 있다. 마찬가지로 다항식 $B(x)$도 위와 같이 표현될 수 있으며, 두 다항식을 곱한 $C(x)$는 6차 다항식으로 총 7개의 계수를 요구한다. 따라서 SABER의 Toom-Cook 알고리즘에서의 평가과정은 7개의 점에 대한 결과값이 요구된다. 각 포인트에 대한 결과값을 구하는 평가과정은 행렬의 곱으로 나타낼 수 있으며, 하나의 다항식에 대한 평가과정은 Fig. 1과 같다.

$$\begin{bmatrix} A(0) \\ A(1) \\ A(-1) \\ 8A(\frac{1}{2}) \\ 8A(-\frac{1}{2}) \\ A(2) \\ A(inf) \end{bmatrix} = \begin{bmatrix} 0^0 & 0^1 & 0^2 & 0^3 \\ 1^0 & 1^1 & 1^2 & 1^3 \\ -1^0 & -1^1 & -1^2 & -1^3 \\ 8(\frac{1}{2})^0 & 8(\frac{1}{2})^1 & 8(\frac{1}{2})^2 & 8(\frac{1}{2})^3 \\ 8(-\frac{1}{2})^0 & 8(-\frac{1}{2})^1 & 8(-\frac{1}{2})^2 & 8(-\frac{1}{2})^3 \\ 2^0 & 2^1 & 2^2 & 2^3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Fig. 1. Evaluation in Toom-Cook Algorithm

2.1.2 점별 곱셈

위의 평가과정을 통해 총 7개 포인트에 대한 결과값을 저장한 두 다항식 A, B에 대해 63차 다항식 7개가 생성된다. 점별 곱셈 과정은 각 포인트에 대한 결과값을 저장하고 있는 두 다항식을 곱해주는 과정이며, 예를 들어 C(0)=A(0)*B(0)를 구하는 과정이다. 이때 Karatsuba-2way를 사용한다. Karatsuba를 사용하면 Operand-Scanning 곱셈을 사용하는 것보다 한 번의 곱셈 연산을 줄일 수 있으며 Karatsuba 곱셈 과정은 Fig. 2와 같다. Karatsuba-2way를 사용하므로 16차 다항식에서의 곱셈은 Operand Scanning 방법을 통해 수행된다. 결과적으로 점별 곱셈 과정은 총 16차 다항식 7X9=63번의 Operand Scanning 연산량을 가지고 있다.

$$A = A_H * 2^n + A_L$$
$$B = B_H * 2^n + B_L$$
$$A * B = A_H B_H * 2^n + [(A_H + A_L)(B_H + B_L) - A_H B_H - A_L B_L] * 2^{n/2} + A_L B_L$$

Fig. 2. Karatsuba Multiplication

2.1.3 보간

점별 곱셈을 수행하여 총 7개 포인트의 곱셈 값에 대한 결과값이 저장된다. 마지막으로 보간과정은 점별 곱셈을 수행한 C(0), C(1), C(-1), 64C(1/2), 64C(-1/2), C(2), C(inf)를 통해 최종적으로 두 다항식 A,B의 곱셈 결과인 C의 계수인 $c_6, c_5, c_4, c_3, c_2, c_1, c_0$를 구하는 과정이다. 위의 과정은 행렬의 곱셈으로 표현할 수 있으며 Fig. 3과 같다. 단순 행렬 곱셈을 수행함으로써, 다항식 C의 7개 계수를 모두 구할 수 있다. 하지만 평가과정에서는 256차 다항식을 64차 다항식 4개로 분할하였지만, 보간과정에서는 256차 다항식을 7개로 분할하므로, 구현상에서 평가과정은 독립적으로 결과값이 배열에 저장되지만, 보간과정은 Overlap 과정과 Accumlator가 요구된다.

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_6 \\ c_5 \end{bmatrix} = \begin{bmatrix} 0^0 & 0^1 & 0^2 & 0^3 & 0^4 & 0^5 & 0^6 \\ 1^0 & 1^1 & 1^2 & 1^3 & 1^4 & 1^5 & 1^6 \\ -1^0 & -1^1 & -1^2 & -1^3 & -1^4 & -1^5 & -1^6 \\ 64(\frac{1}{2})^0 & 64(\frac{1}{2})^1 & 64(\frac{1}{2})^2 & 64(\frac{1}{2})^3 & 64(\frac{1}{2})^4 & 64(\frac{1}{2})^5 & 64(\frac{1}{2})^6 \\ 64(-\frac{1}{2})^0 & 64(-\frac{1}{2})^1 & 64(-\frac{1}{2})^2 & 64(-\frac{1}{2})^3 & 64(-\frac{1}{2})^4 & 64(-\frac{1}{2})^5 & 64(-\frac{1}{2})^6 \\ 2^0 & 2^1 & 2^2 & 2^3 & 2^4 & 2^5 & 2^6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} c(0) \\ c(1) \\ c(-1) \\ 64c(\frac{1}{2}) \\ 64c(-\frac{1}{2}) \\ c(2) \\ c(inf) \end{bmatrix}$$

Fig. 3. Interpolation in Toom-Cook Algorithm

2.2 ARMv8 Microcontroller

ARMv8 A Series는 64-bit ARM 프로세서와 병렬처리가 가능한 NEON 엔진을 지원한다. 64-bit ARM 프로세서에서는 배럴 시프터와 x0-x30까지의 64-bit 레지스터를 지원하며, ARM에서 사용할 수 있는 A64 명령어 집합[6]을 지원한다. NEON에서는 병렬처리를 수행할 수 있는 v0-v31까지의 128-bit 벡터 레지스터와 ASIMD 명령어 집합 [7]을 지원한다. 128-bit 벡터 레지스터 내에서는 8-bit, 16-bit, 32-bit, 64-bit 단위로 핸들링이 가능하

Table 1. Summary of A64 and ASIMD Instruction Set in ARMv8-A Series [6], [7]

A64 Instructions			
Instruction	Operand	Description	Cycles
LDR, STR	xd, [xn]	Loading the data from memory to general-purpose register, and storing the data from general-purpose register to memory, [xn]→xd, xd→[xn]	3
ADD, SUB	xd, xm, xn	Addition and subtraction, xd=xm+xn, xd=xm-xn	1
AND	xd, xm, xn	AND operation, xd=xm&xn	1
ASIMD Instructions			
Instruction	Operand	Description	Cycles
LD1, ST1	{vn-vm}, [xd]	Loading the data from memory to vector registers, and storing the data from vector registers to memory, [xd]→{vn-vm}, {vn-vm}→{xd}	4
ADD, SUB	vn, vm, vd	Vector addition and subtraction, vn=vm+cd, vn=vm-vd	1
USHR	vn, vm, #im	Vector left shift operation, vn=(vm,#im)	1
USHL	vn, vm, vd	Vector right shift operation, vn=(vm,#vd)	1
MUL	vn, vm, vd	Vector multiplication, vn=vm*vd	2

다. 게다가 ARM/NEON은 독립적인 모듈로 연산을 수행하여 ARM과 NEON의 각 구현을 단순히 합치게 되면 ARM과 NEON의 실행시간의 합이지만, ARM/NEON 프로세서의 인터리빙 적인 접근은 ARM 연산들의 지연시간을 NEON 오버헤드로 숨길 수 있어 효과적으로 성능향상이 가능하다. Table. 1은 본 논문에서 사용하는 A64, ASIMD 명령어들의 설명과 Clock Cycles를 나타낸다.

III. Related Work

본 장에서는 임베디드 환경에서 양자내성암호의 연구결과를 소개한다. AFRICA CRYPTO 2019 [8]에서는 ARM Cortex-M4 환경에서 KYBER에 대한 메모리 최적화 및 고속 구현을 제시하였다. 특히 KYBER에서 핵심 연산인 NTT(Number Transform Theory)에 대해 타겟 장비에서 지원하는 벡터 DSP 명령어를 사용한 최적화 기술을 소개하였다. 게다가 메모리 최적화를 적용하여 SABER가 요구하는 메모리 사용량의 절반만을 필요로 하였으며, 제안한 모든 구현이 Constant-time이므로 타이밍 공격에 대한 안전성을 제공하였다.

CHES 2020 [9]에서는 ARM Cortex-M4와 AVX2 환경에서 Toom-Cook 알고리즘의 최적화 구현 연구가 소개되었다. Toom-Cook 알고리즘에서 Lazy 보간과 Pre-computation 최적화 방법을 제안하였다. 이를 임베디드 환경인 ARM Cortex-M4에 적용하고 16차 다항식 곱셈에 대한 메모리 최적화 방법도 제시하였다. 결과적으로 이전 연구보다 메모리 사용량을 2.6KB에서 5.7KB까지 효율적으로 감소시켰다.

CHES 2021 [10]에서는 ARM Cortex-M3, ARM Cortex-M4에서 Crystal-Dilithium의 메모리 최적화 및 고속 구현을 제안하였다. Crystal-Dilithium의 핵심 연산인 NTT 연산을 최적화하였으며, NTT, NTT^-1에서 20% 성능을 향상시켜 전체적으로 이전 연구보다 KeyGen에서 7%, Sign에서 15%, Verify에서 9% 성능 향상을 달성하였다.

임베디드 환경에서 양자내성암호의 최적화 구현 연구들은 대부분 ARM Cortex-M4 환경에서 최적화를 수행하였다. 하지만 최근 ARM Cortex-A Series는 모바일, 태블릿, 자율주행에서 널리 사용되고 더욱 강력한 NEON 엔진까지 지원하여 다양한 암호 최적화 연구가 활발히 수행되고 있다. 본 논

문에서는 최초로 ARM Cortex-A Series(ARMv8)에서 ARM/NEON 프로세서를 활용한 Toom-Cook 알고리즘의 평가 및 보간 단계의 최적화 구현 방법을 소개한다.

Table 2. Summary of Register Scheduling

Register Scheduling	
Register	Description
Load	
x0, x5, x6, x7	Storing addresses of a3, a2, a1, a0 with 64 coefficients in polynomial A
Store	
x2, x8, x9, x10, x11, x12, x13	Storing addresses for the result values of A(0), A(1), A(-1), 8A(1/2), 8A(-1/2), A(2), A(inf)
Coefficient	
x13, x14, x15, x16	Storing 4 coefficients each in a3, a2, a1, and a0
v0, v1, v2, v3	Storing 8 coefficients each in a3, a2, a1, and a0
Q	
x29, v30.8h	Storing Q to apply modulus

IV. Toom-Cook 알고리즘 최적화

4.1 평가

본 절에서는 Toom-Cook 알고리즘의 평가과정에 대한 ARM/NEON 프로세서를 활용한 최적화 방법을 소개한다. 먼저 ARM 프로세서에서의 병렬 최적화 방법을 소개하고 그다음 NEON 엔진에서의 병렬 최적화 방법을 소개한다. 마지막으로 ARM/NEON의 각 구현을 인터리빙하여 효율적으로 ARM 명령어에 대한 지연시간을 NEON 명령어로 숨기는 방법을 소개한다. Table 2는 ARM/NEON 프로세서의 각 구현에서 레지스터 스케줄링을 보여준다.

4.1.1 Optimization in ARM Processor

ARM Cortex-A series에서 64-bit ARM 프로세서는 병렬처리를 지원하지 않는다. 하지만 Toom-Cook 알고리즘의 평가과정에서는 2의 파워승 곱셈, 덧셈, 뺄셈 연산으로 구성되어있다. 또한 SABER의 모듈러스 Q는 2^{13}이다. 따라서 우리는 64-bit ARM 프로세서에서 4개 계수에 대한 병렬처리 방법을 소개한다.

제안하는 ARM 프로세서에서 Toom-Cook 알고리즘 평가과정의 전체과정은 Fig. 4와 같다. 1~4단계는 a3, a2, a1, a0에서 각각 4개의 계수를 ARM 레지스터에 로드한다. 5~13단계는 A(1), A(-1)을 구하는 과정이다. 이때 뺄셈에 경우에는 Borrow가 발생할 수 있어 병렬적으로 처리하기 위해서는 A-B가 아닌 P+A-B로 계산하여 발생할 수 있는 Borrow를 제거한다.

14~16단계는 계산이 완료된 A(1), A(-1) 값을 메모리에 저장한다. 17~26단계는 8A(1/2), 8A(-1/2)를 계산하는 과정이다. 이때 2의 파워 승 곱셈은 배럴 시프터를 사용하여 연산에 대한 클럭 사이클을 소모하지 않는다. 마찬가지로 뺄셈에서는 Borrow를 제거하기 위해 A-B가 아닌 P+A-B로 연산을 처리한다. 이를 통해 병렬적으로 뺄셈이 가능하다. 27~28단계는 8A(1/2), 8A(-1/2)의 결과값을 메모리에 저장한다.

29~33단계는 A(2)를 계산하는 과정이며, 마찬가지로 2의 파워승 곱셈에서는 배럴 시프터를 사용한다. 마지막으로 34~36단계에서는 A(2)와 A(inf) 값을 메모리에 저장한다. 위의 과정을 통해 a3, a2, a1, a0의 다항식에서 4개의 계수에 대한 Toom-Cook 알고리즘의 평가과정이 처리된다.

Algorithm 1 Parallel Implementation of Evaluation of Toom-Cook in ARM Processor

```
Require: x13, x14, x15, x16
Ensure: A(0), A(1), A(-1), 8A(1/2), 8A(-1/2), A(2), A(inf)
    Loading 4 coefficient in polynomial A
 1: LDR x13, [x1], #8
 2: LDR x14, [x5], #8
 3: LDR x15, [x6], #8
 4: LDR x16, [x7], #8
    Operating A(1), A(-1)
 5: ADD x17, x13, x15
 6: ADD x18, x14, x16
 7: ADD x19, x17, x18
 8: AND x17, x17, x29
 9: AND x18, x18, x29
10: ADD x20, x26, x17
11: SUB x20, x20, x18
12: AND x19, x19, x29
13: AND x20, x20, x29
    Storing A(0), A(1), A(-1)
14: STR x13, [x3], #8
15: STR x19, [x8], #8
16: STR x20, [x9], #8
    Operating 8A(1/2), 8A(-1/2)
17: ADD x17, x15, x15
18: ADD x18, x17, x13, LSL #3
19: ADD x19, x16, x14, LSL #2
20: AND x18, x18, x29
21: AND x19, x19, x29
22: ADD x20, x18, x19
23: ADD x18, x18, x26
24: SUB x21, x18, x19
25: AND x20, x20, x29
26: AND x21, x21, x29
    Storing 8A(1/2), 8A(-1/2)
27: STR x20, [x10], #8
28: STR x21, [x11], #8
    Operating A(2)
29: ADD x17, x14, x14
30: ADD x17, x17, x16, LSL #3
31: ADD x17, x17, x15, LSL #2
32: ADD x17, x17, x13
33: AND x17, x17, x29
    Storing A(2), A(inf)
34: STR x17, [x12], #8
35: AND x16, x16, x29
36: STR x16, [x25], #8
```

Fig. 4. Evaluation Implementation in ARM Processor

4.1.2 Optimization in NEON Engine

NEON 엔진에서는 128-bit 내에서 8-bit, 16-bit, 32-bit, 64-bit 단위로 병렬처리가 가능하고, SABER의 모듈러스 $Q=2^{13}$이므로 8개 계수에 대한 병렬처리 방법을 제안한다. NEON 엔진을 활용한 Toom-Cook 알고리즘의 평가과정의 전체과정은 Fig. 5와 같다. 16-bit 단위로 병렬처리를 하므로 lane은 8h로 계산된다.

1~4단계는 a3, a2, a1, a0에서 각각 8개의 계수를 벡터 레지스터들에 로드한다. 5~10단계는 A(1), A(-1)을 계산하는 과정으로, ARM 프로세서와 달리 NEON 엔진에서는 자동으로 병렬처리가 가능하므로 Borrow에 대해 고려할 필요가 없다. 11~13단계는 계산이 완료된 A(1), A(-1) 값을 메모리에 로드한다.

14~22단계에서는 8A(1/2), 8A(-1/2)를 계산하는 과정으로, v27, v28, v29에는 3, 2, 1이 16-bit 단위로 채워져 있다. USHL 명령어를 통해 병렬적으로 8, 4, 2의 곱셈이 처리된다. 23~24단계에서는 8A(1/2), 8A(-1/2)의 결과값이 메모리에 저장된다. 25~31단계는 A(2)를 계산하는 과정이다. 마찬가지로 USHL 명령어를 통해 병렬적으로 2의 파워승 곱셈이 수행된다. 마지막으로 32~34단계는 8A(2), A(inf)의 결과값이 메모리에 저장된다. 위의 과정을 통해 a3, a2, a1, a0의 64개의 계수를 갖는 다항식에서 8개 계수에 대한 평가과정 처리가 완료된다.

Algorithm 2 Parallel Implementation of Evaluation of Toom-Cook in NEON Engine

Require: v0,v1,v2,v3
Ensure: A(0), A(1), A(-1), 8A(1/2), 8A(-1/2), A(2), A(inf)

Loading 8 coefficient in polynomial A
1: LD1 {v0.8h}, [x0], #16
2: LD1 {v1.8h}, [x5], #16
3: LD1 {v2.8h}, [x6], #16
4: LD1 {v3.8h}, [x7], #16

Operating A(1), A(-1)
5: ADD v4.8h,v0.8h,v2.8h
6: ADD v5.8h,v1.8h,v3.8h
7: ADD v6.8h,v4.8h,v5.8h
8: SUB v7.8h,v4.8h,v5.8h
9: AND v6.16b,v6.16b,v30.16b
10: AND v7.16b,v7.16b,v30.16b

Storing A(0), A(1), A(-1)
11: ST1 {v0.8h}, [x2], #16
12: ST1 {v6.8h}, [x8], #16
13: ST1 {v7.8h}, [x9], #16

Operating 8A(1/2), 8A(-1/2)
14: USHL v4.8h,v0.8h,v29.8h
15: ADD v5.8h,v4.8h,v2.8h
16: USHL v4.8h,v5.8h,v28.8h
17: USHL v5.8h,v1.8h,v29.8h
18: ADD v5.8h,v5.8h,v3.8h
19: ADD v6.8h,v4.8h,v5.8h
20: SUB v7.8h,v4.8h,v5.8h
21: AND v6.16b,v6.16b,v30.16b
22: AND v7.16b,v7.16b,v30.16b

Storing 8A(1/2), 8A(-1/2)
23: ST1 {v6.8h}, [x10], #16
24: ST1 {v7.8h}, [x11], #16

Operating A(2)
25: USHL v4.8h,v3.8h,v27.8h
26: USHL v5.8h,v2.8h,v29.8h
27: USHL v6.8h,v1.8h,v28.8h
28: ADD v7.8h,v4.8h,v5.8h
29: ADD v7.8h,v7.8h,v6.8h
30: ADD v7.8h,v7.8h,v0.8h
31: AND v7.16b,v7.16b,v30.16b

Storing A(2), A(inf)
32: ST1 {v7.8h}, [x12], #16
33: AND v3.16b,v3.16b,v30.16b
34: ST1 {v3.8h}, [x13], #16

Fig. 5. Evaluation Implementation in NEON Engine

4.1.3 Optimization in ARM/NEON Processor

우리는 평가과정에서 ARM/NEON의 인터리빙 구현을 제안한다. ARM/NEON의 인터리빙 구현은 Fig. 6와 같으며, 인터리빙 구현을 통해 ARM 계산에 대한 지연시간이 NEON 오버헤드로 효율적으로 숨겨진다. ARM에서는 4개의 계수에 대한 평가과정에 대해 병렬 처리하였으며, NEON에서는 8개의 계수에 대한 평가과정에 대해 병렬처리를 하였다. 우리는 인터리빙 구현을 통해 12개의 계수에 대해 병렬처리가 수행되며, 이때 ARM에서 처리되는 4개의 계수에 대한 평가과정에서의 연산 지연시간이 NEON으로 숨겨져 효과적으로 성능이 향상된다. 위의 과정을 5번 반복하고 마지막은 4개의 계수에 대해 ARM에서 평가과정을 처리하게 되면 64개의 계수를 갖는 a3, a2, a1, a0의 평가과정이 완료된다. 마찬가지로 다항식 B(x)도 위와 같은 과정을 통해 평가과정을 수행한다.

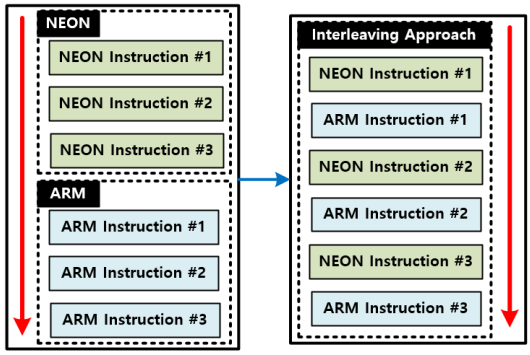


Fig. 6. Interleaving Approach

4.2 보간

보간과정에서는 NEON을 활용하여 8개의 계수를 동시에 처리하고 메모리 접근을 최소화하는 구현 방법을 소개한다. 제안하는 보간과정의 구현을 위한 레지스터 스케쥴링은 Fig. 7과 같다. x10~x16, x20~x26은 값을 로드하기 위해 주소값을 저장하고 있는 레지스터이며, x1~x8은 결과값을 저장하기 위한 주소 값을 저장하고 있는 레지스터이다.

v0-v13은 각 8개의 계수를 저장하기 위한 벡터 레지스터이며, 나머지는 변수값을 저장하고 있는 레지스터이다. 기존 레퍼런스 구현에서는 for문을 돌면

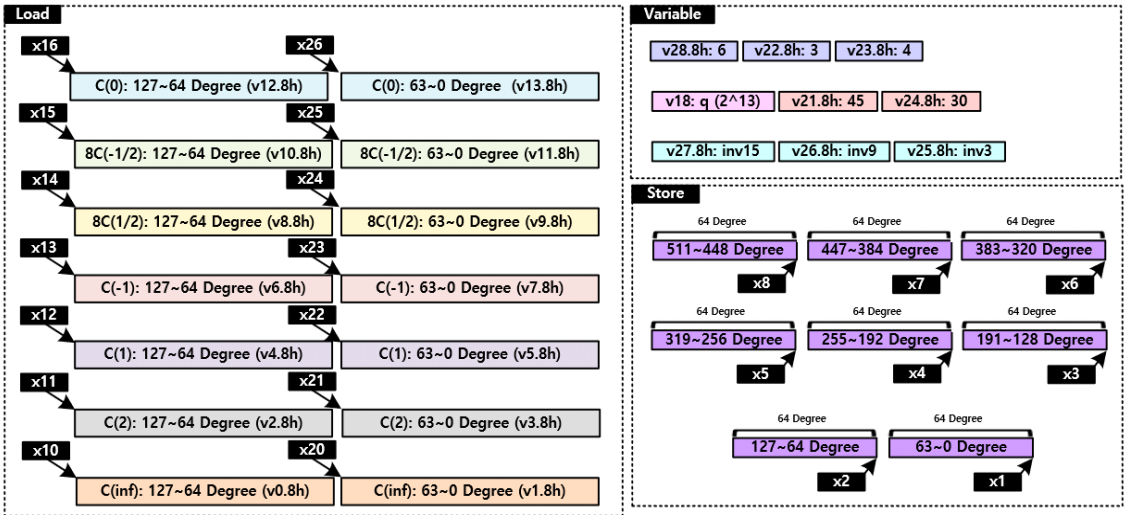


Fig. 7. Register Scheduling for Interpolation in Toom-Cook Algorithm

서 0~63까지 수행하여 0~447차의 계수의 결과값을 계산하고 64~127까지 수행하여 64~511차의 계수의 결과값을 계산하여 두 값을 더함으로써 최종 결과값이 반환된다. 하지만 임베디드 환경에서는 모든 값을 레지스터 내에 임시 저장을 할 수 없으므로, 위와 같은 구현은 많은 메모리 사용량을 요구한다.

따라서 우리는 임베디드 환경에서 메모리 사용량을 최소화하는 구현을 제시한다. 제안하는 방법은 Fig. 8과 같다. 128차 결과값인 C(0) … C(inf)를 64개의 계수로 나누어 행렬 곱셈을 처리하고 바로 그 결과값들을 더하여 최종 결과값을 구함으로써 Accumulaotr에 대한 메모리 접근 부하를 최소화하였다. 제안하는 구현은 레퍼런스 구현과 비교하여 320번의 메모리 사용을 줄일 수 있다.

제안하는 구현의 코드는 Fig. 9와 같다. 1~35단계는 레퍼런스 구현에서의 for 64~127단계에서 8

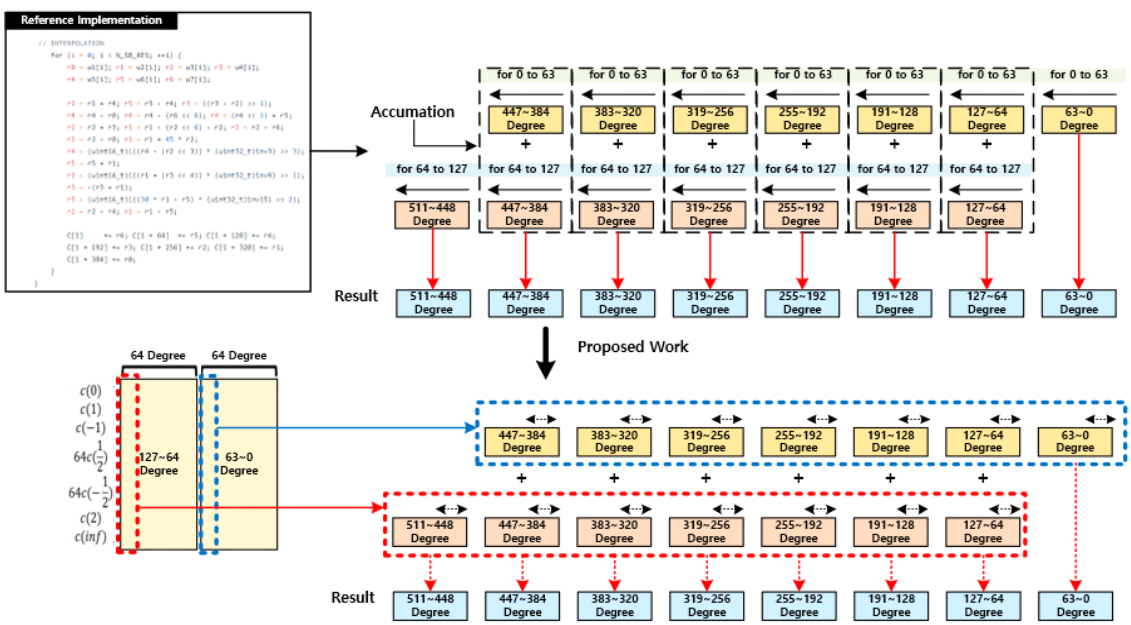


Fig. 8. Proposed Optimization of Interpolation in Toom-Cook Algorithm

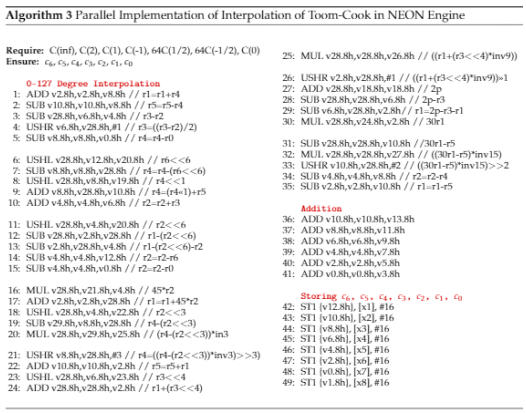


Fig. 9. Interpolation Implementation in NEON Engine

개의 계수에 대한 보간과정을 병렬처리하는 단계이다. 1~35단계와 마찬가지로 for 0~63단계 에 대한 8개 계수에 대한 보간과정을 병렬처리 후, 36~41단계를 통해 바로 두 개의 결과값에 대해 덧셈을 수행함으로써, 추가적인 메모리 접근 없이 최종 결과값이 반환된다. 42~49 단계는 보간의 최종 결과값에 대해 메모리에 저장하는 단계이다. 위는 8개의 계수에 대한 병렬처리 과정으로 총 8번 위의 과정을 반복하게 되면 보간과정이 완료된다.

V. 성능 측정

본 절에서는 제안하는 Toom-Cook 알고리즘의 평가 및 보간과정에 대한 성능 측정 결과를 제안한다. 우리는 라즈베리파이 4B 환경에서 성능을 측정하였으며, 라즈베리파이 4B는 4개의 코어 ARM Cortex-A72(ARMv8)과 4GB 내장 메모리를 탑재하고 있다. 우리의 타겟장비는 우분투 20.04 운영체제가 설치되어있으며, VS Codium 개발환경 내 Aarch64-gnu-gcc 컴파일러를 통해 성능을 측정하였다. 또한, ARMv8 환경에서 이전 연구결과가 존재하지 않아 레퍼런스 구현과의 성능을 비교하였으며, 레퍼런스 구현은 C로 구현되어있고 사이클 측정 결과는 -O2 최적화 옵션을 적용하였다.

성능 측정 결과는 Table 3과 같다. 평가과정은 두 개의 다항식 A, B에 관한 결과의 클럭 사이클을 측정하였다. 평가과정에서는 ARM/NEON 인터리빙 구현과 8개 계수에 대한 병렬처리 기법을 통해 레퍼런스 구현보다 약 3.5배 향상되었다. 게다가 보간과정에서는 8개 계수에 대한 병렬처리 기법과 메모리 접근 부하를 최소화하여 레퍼런스 구현보다 약 5배 향상된 결과를 달성하였다. SABER에서 가장 성능 부하가 큰 부분은 행렬 곱셈으로 제안하는 방법을 통해 SABER 전체적으로는 더 큰 성능향상이 기대된다.

Table 3. Comparison of performance between reference implementation and our work

	Cycles
Reference (Evaluation)	2,227
Reference (Interpolation)	3,121
Our Work (Evaluation)	602
Our Work (Interpolation)	651

VI. 결 론

양자 컴퓨팅 환경의 발전으로, 기존 공개키 암호의 안전성이 깨지면서, 양자내성암호의 중요성이 강조되고 있다. 특히 양자내성암호 중 격자기반 암호는 빠른 속도를 제공하여 실제 임베디드 환경에서도 적용하다. 게다가 ARMv8은 저전력과 고효율의 장점으로 앞으로도 임베디드 SoC로서 널리 활용될 것으로 기대된다. 하지만 임베디드 환경에서 양자내성암호의 최적화 구현연구 대부분은 ARM Cortex-M4 환경에서 수행하였다. 본 논문에서는 최초로 ARMv8 환경에서 NIST PQC SABER의 핵심 연산인 다항식 곱셈 Toom-Cook 알고리즘에 대한 최적화 구현 방법을 제안하였다. 제안하는 방법을 통해 평가 및 보간 과정에서 레퍼런스 구현보다 3.5배, 5배의 성능 향상을 달성하였다. 향후 Toom-Cook 알고리즘의 Karatsuba 곱셈 및 NTT 곱셈에 대한 최적화 구현을 연구할 계획이다.

References

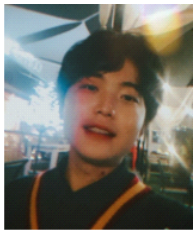
[1] Peter W. Shor, "Polynomial-time algorithm for prime factorization and discrete logarithms on a quatum computer", SIAM review, vol. 41, no. 2, pp. 303-332, Oct. 1997

[2] NIST PQC Standardization "https://csrc.nist.gov/projects/post-quantum-crypt

ography", 2021. 4. 15
〔3〕 Hwajeong Seo, Taehwan Park, Shinwook Heo, Gyuwon Seo, Bongjin Bae, Zhi Hu, Lu Zhou, Yasuyuki Nogami, Youwen Zhu, Howon Kim, "Parallel Implementations of LEA, Revisited", WISA, 10144, pp. 318-330, Aug. 2016
〔4〕 Hwajeong Seo, Kyuhwang An, Hyeokdong Kwon, Taehwan Park, Zhi Hu and Howon Kim, "Parallel Implementations of CHAM", WISA, 11402, pp. 93-104, Aug. 2018
〔5〕 Hwajeong Seo, Taehwan Park, Janghyun Ji, Zhi Hu, and Howon Kim, "ARM/NEON Co-design of Multiplication/Squaring", WISA, 10763, pp. 72-84, Aug. 2017
〔6〕 ARMv8 A64 Instruction set, "https://developer.arm.com/documentation/ddi0596/2020-12/Base-Instructions", 2021. 4. 15
〔7〕 ARMv8 ASIMD Instruction set, "https://developer.arm.com/documentation/ddi0596/2020-12/SIMD-FP-Instructions", 2021. 4. 15
〔8〕 Leon Botros, Matthias J. Kannwischer, Peter Schwabe, "Memory-Efficient High-Speed Implementation of Kyber on Cortex-M4", AFRICACRYPTO, vol. 11627, pp 209−228, July. 2019
〔9〕 Jose Maria Bermudo Mera, Angshuman Karmakar, and Ingrid Verbauwhede, ""Time-memory trade-off in Toom-Cook multiplication: an application to module-lattice based cryptography", CHES, vol. 2020, no. 2, pp. 222-244, Sep. 2020,
〔10〕 Denisa O. C. Greconici, Matthias J. Kannwischer and Daan Sprenkels, "Compact Dilithium Implementations on Cortex-M3 and Cortex-M4", CHES, vol. 2021, no. 1, pp. 1-24, Sep. 2021

〈저자소개〉

송 진 교 (JinGyo Song) 학생회원
2020년: 국민대학교 정보보안암호수학과 졸업
2020년~현재: 국민대학교 금융정보보안학과 석사과정
〈관심분야〉 암호최적화, 임베디드 보안, 양자내성암호



김 영 범 (YoungBeom Kim) 학생회원
2021년: 국민대학교 정보보안암호수학과 졸업
2021년~현재: 국민대학교 금융정보보안학과 석사과정
〈관심분야〉 암호모듈검증, 양자내성암호

서 석 충 (Seog Chung Seo) 정회원
2011년 8월: 고려대학교 정보보호대학원 박사
2013년 11월: 삼성전자 종합기술원 전문연구원
2014년 4월: 삼성전자 DMC 연구소 책임연구원
2019년 2월: 국가보안기술연구소 선임연구원
2019년 3월~현재: 국민대학교 정보보안암호수학과 조교수
〈관심분야〉 암호최적화, 공개키 암호, 암호모듈검증,네트워크보안