Optimized Software Implementations of CRYSTALS-Kyber, NTRU, and Saber Using NEON-Based Special Instructions of ARMv8

Cryptographic Engineering Research Group @ George Mason University

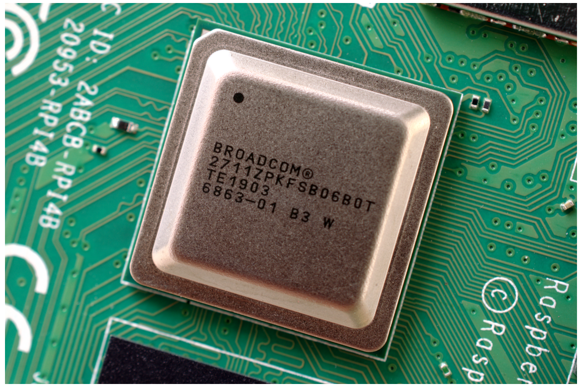
Duc Tri Nguyen and Kris Gaj

Introduction

- NEON is an alternative name for Advanced Single Instruction Multiple Data (ASIMD) extension to the ARM Instruction Set Architecture, mandatory since ARMv7-A.

- NEON provides 32x**128-bit** vector registers. Compared with Single Instruction Single Data (SISD), ASIMD can have ideal speed-up in the range 2..16 (for 64..8-bit operands).

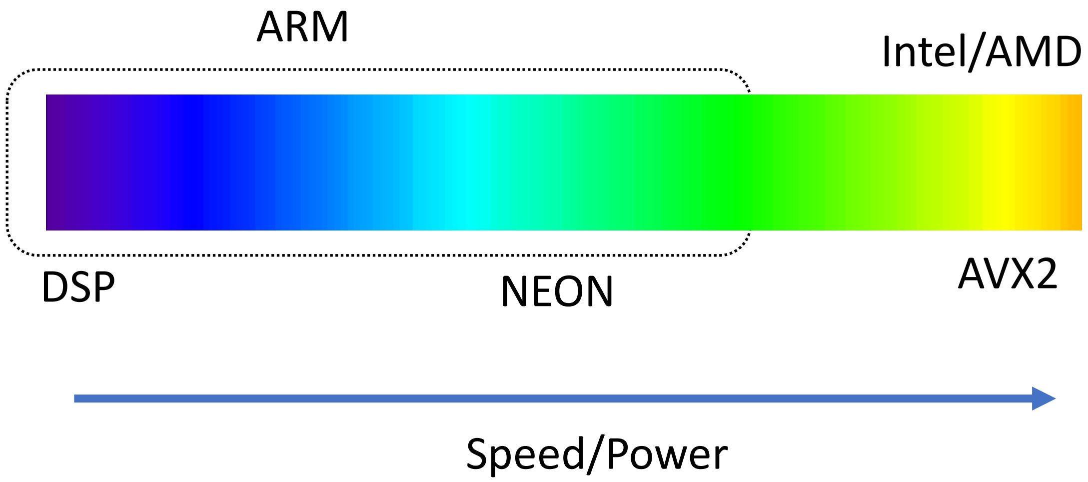
Apple M1:
part of new MacBook Air, MacBook Pro, Mac Mini, iMac, and iPad Pro



Broadcom SoC, BCM2711:
part of the Raspberry Pi 4 single-board computer

Introduction

- Most software implementations of PQC candidates on:
  - Intel/AMD (w/ AVX2 extension)
  - Cortex-M4 (w/ DSP extension)[1]
- Lack of NEON implementations on ARMv7 and ARMv8 architectures

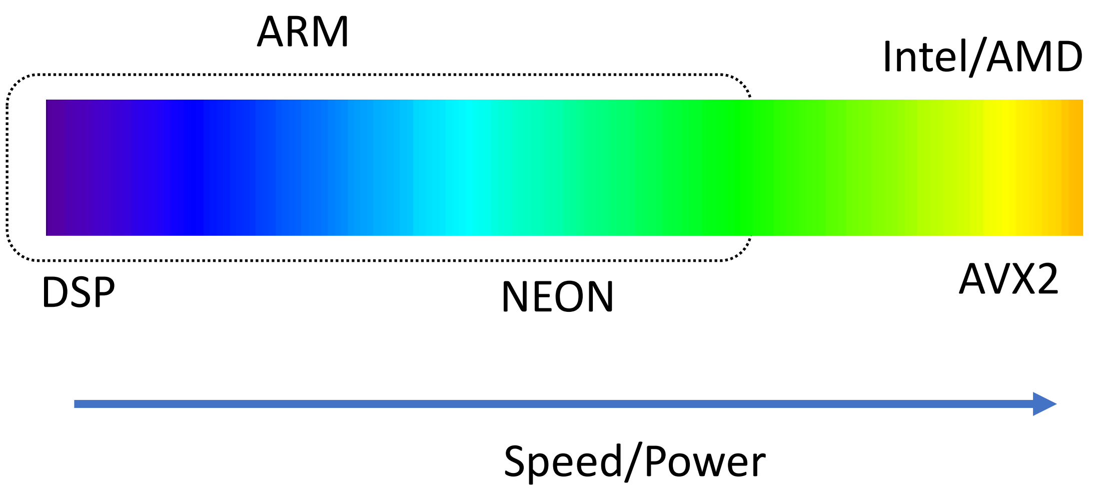


[1] M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen, pqm4 - Post-quantum crypto library for the ARM Cortex-M4, https://github.com/mupq/pqm4

Introduction

- Our goal is to fill the gap between low-power embedded processors and high-performance x86-64 platforms.

- We developed constant-time, optimized ARMv8 implementations of 3 KEM finalists:
  - CRYSTALS-Kyber
  - NTRU
  - Saber



Polynomial Multiplication

$$Schoolbook \longrightarrow Toom-Cook \longrightarrow NTT$$

$$\uparrow \qquad\qquad \uparrow \qquad\qquad \uparrow$$

$$O(n^2) \qquad\qquad O(n^{\frac{\log(2k-1)}{\log k}}) \qquad\qquad O(n \log n)$$

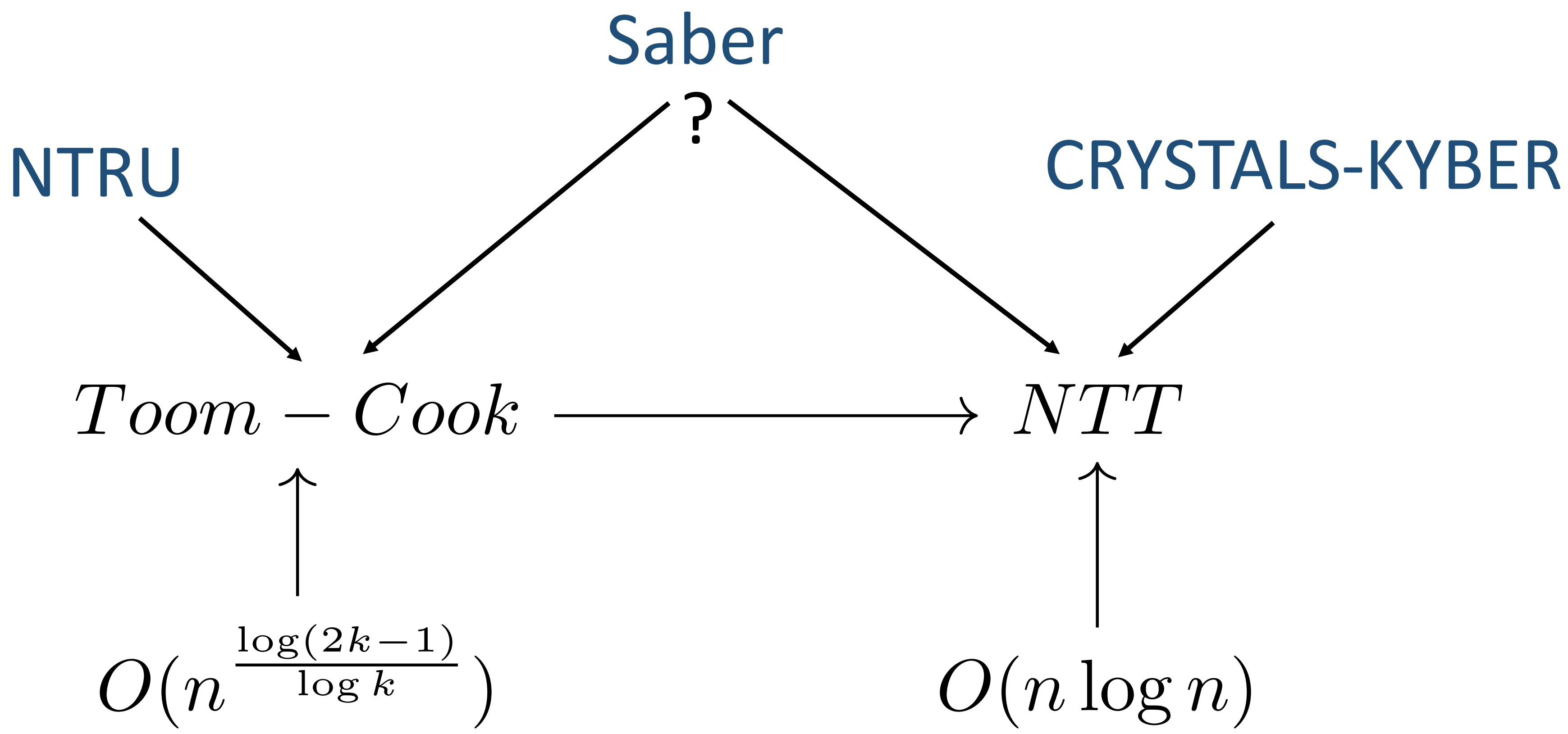
Typically:

k=2: Karatsuba : $O(n^{1.58})$

k=3: Toom-3 : $O(n^{1.46})$

k=4: Toom-4 : $O(n^{1.40})$

Optimal Choice of Algorithms



Based on the analysis of algorithms, their parameters, and AVX2 implementations for the 3 lattice-based KEMs finalists

5 Steps of Toom-4

1. Splitting

$$A(x) = x^{\frac{3n}{4}} \sum_{i=\frac{3n}{4}}^{n-1} a_i x^{(i-\frac{3n}{4})} + \cdots + x^{\frac{n}{4}} \sum_{i=\frac{n}{4}}^{\frac{2n}{4}-1} a_i x^{(i-\frac{n}{4})} + \sum_{i=0}^{\frac{n}{4}-1} a_i x^i$$

$$= \alpha_3 \cdot x^{\frac{3n}{4}} + \alpha_2 \cdot x^{\frac{2n}{4}} + \alpha_1 \cdot x^{\frac{n}{4}} + \alpha_0$$

2. Evaluation

$$\begin{bmatrix} \mathcal{A}(0) \\ \mathcal{A}(1) \\ \mathcal{A}(\text{-}1) \\ \mathcal{A}(\frac{1}{2}) \\ \mathcal{A}(\text{-}\frac{1}{2}) \\ \mathcal{A}(2) \\ \mathcal{A}(\infty) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ \text{-}1 & 1 & \text{-}1 & 1 \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{2} & 1 \\ \text{-}\frac{1}{8} & \frac{1}{4} & \text{-}\frac{1}{2} & 1 \\ 8 & 4 & 2 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \alpha_3 \\ \alpha_2 \\ \alpha_1 \\ \alpha_0 \end{bmatrix}$$

3. Pointwise multiplication

$$\begin{bmatrix} \mathcal{C}(0) \\ \mathcal{C}(1) \\ \mathcal{C}(\text{-}1) \\ \mathcal{C}(\frac{1}{2}) \\ \mathcal{C}(\text{-}\frac{1}{2}) \\ \mathcal{C}(2) \\ \mathcal{C}(\infty) \end{bmatrix} = \begin{bmatrix} \mathcal{A}(0) \\ \mathcal{A}(1) \\ \mathcal{A}(\text{-}1) \\ \mathcal{A}(\frac{1}{2}) \\ \mathcal{A}(\text{-}\frac{1}{2}) \\ \mathcal{A}(2) \\ \mathcal{A}(\infty) \end{bmatrix} \cdot \begin{bmatrix} \mathcal{B}(0) \\ \mathcal{B}(1) \\ \mathcal{B}(\text{-}1) \\ \mathcal{B}(\frac{1}{2}) \\ \mathcal{B}(\text{-}\frac{1}{2}) \\ \mathcal{B}(2) \\ \mathcal{B}(\infty) \end{bmatrix}$$

5 Steps of Toom-4

4. Interpolation

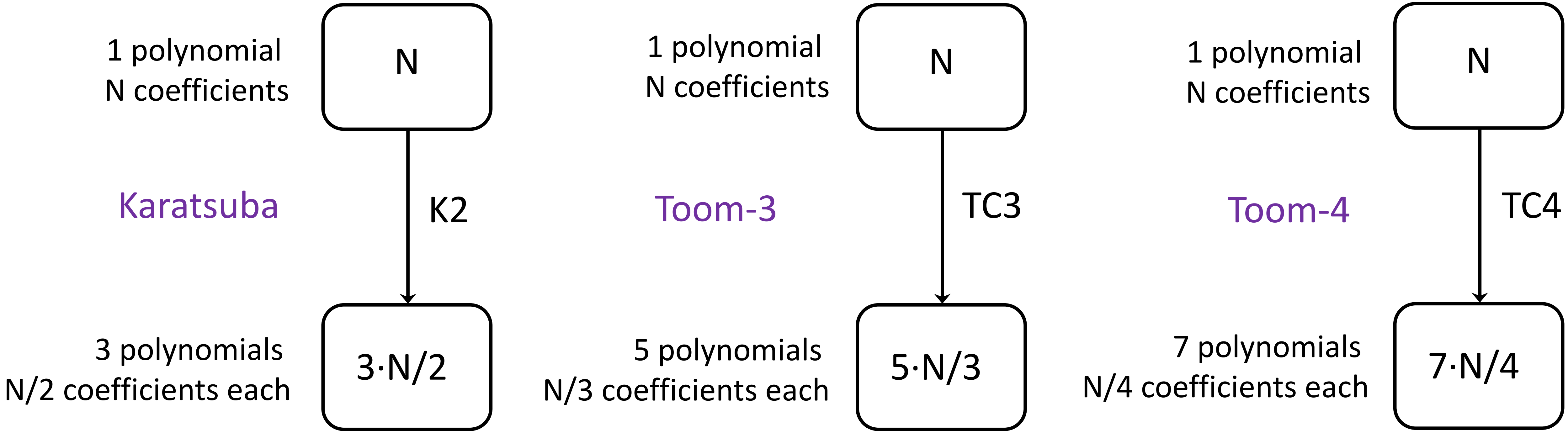
$$\begin{bmatrix}\theta_0\\\theta_1\\\theta_2\\\theta_3\\\theta_4\\\theta_5\\\theta_6\end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 \\ \frac{1}{64} & \frac{1}{32} & \frac{1}{16} & \frac{1}{8} & \frac{1}{4} & \frac{1}{2} & 1 \\ \frac{1}{64} & -\frac{1}{32} & \frac{1}{16} & -\frac{1}{8} & \frac{1}{4} & -\frac{1}{2} & 1 \\ 64 & 32 & 16 & 8 & 4 & 2 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^{-1} \cdot \begin{bmatrix}\mathcal{C}(0)\\\mathcal{C}(1)\\\mathcal{C}(-1)\\\mathcal{C}(\frac{1}{2})\\\mathcal{C}(-\frac{1}{2})\\\mathcal{C}(2)\\\mathcal{C}(\infty)\end{bmatrix}$$

where $\mathcal{C}(\mathcal{X}) = \sum_{i=0}^{6} \theta_i \mathcal{X}^i$

5. Merging

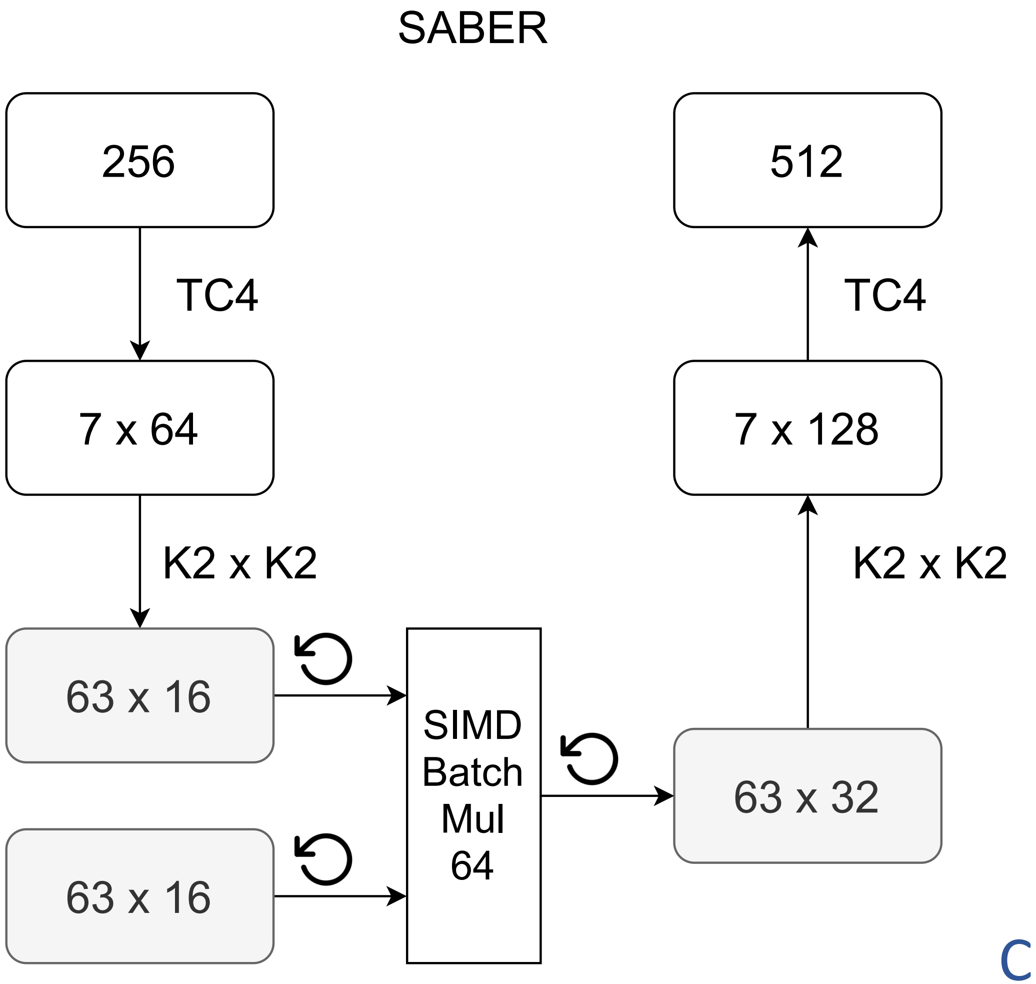
$$C(x) = \sum_{i=0}^{2n-1} a_i x^i = \sum_{i=0}^{6} \theta_i x^{in/4}$$

Toom-Cook: Splitting & Evaluation



Toom-Cook Implementation: Saber

Splitting
& Evaluation



Interpolation
& Merging

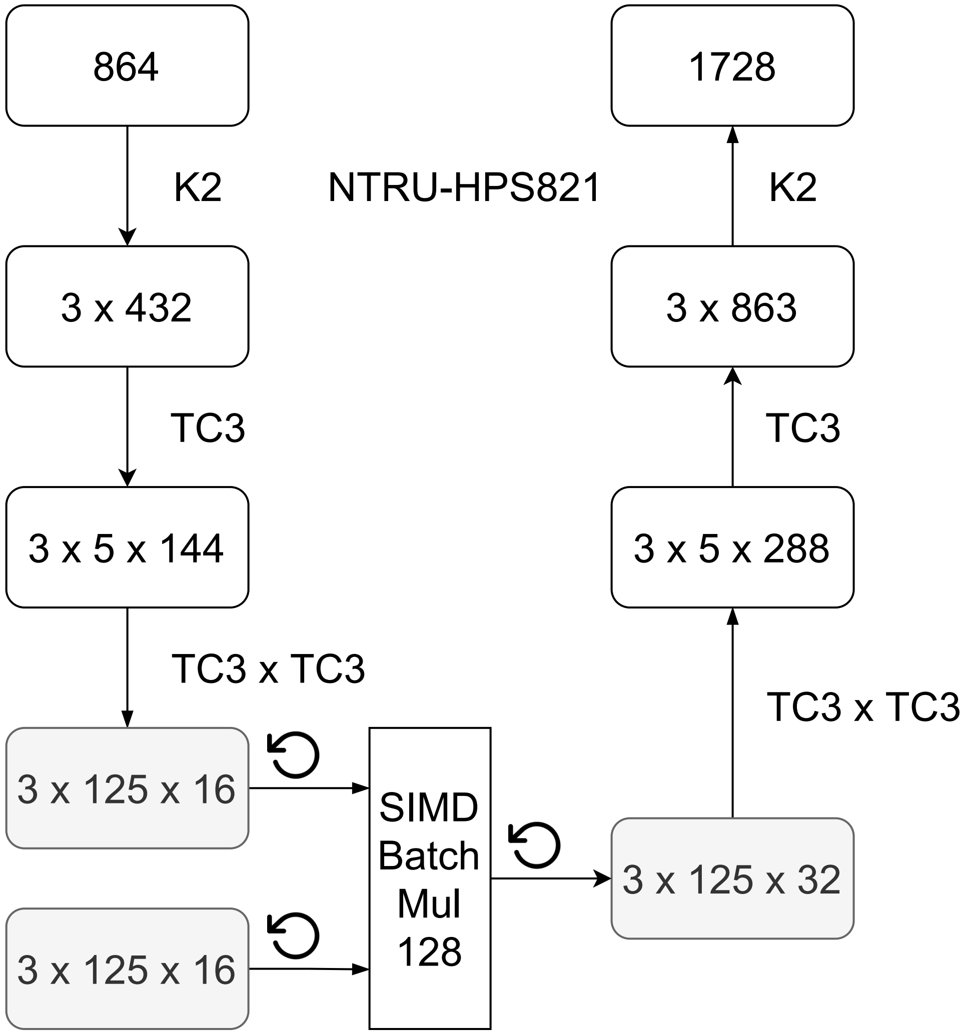
Repeated for A(x) & B(x)

C(x)

Schoolbook 16 x 16: Pointwise Multiplication

Toom-Cook Implementation: NTRU-HPS821

Splitting
& Evaluation



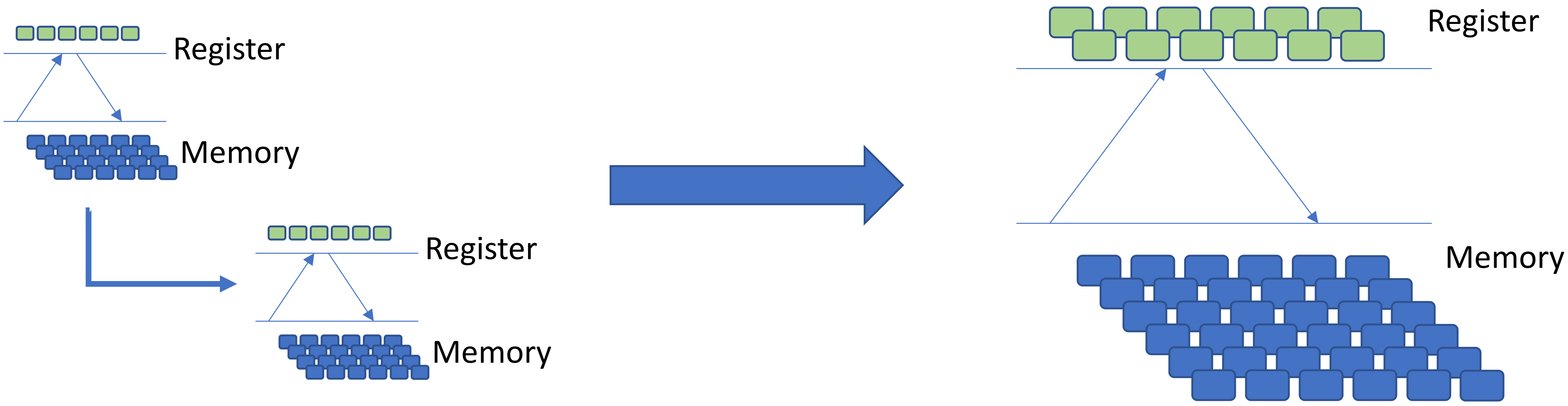
Interpolation
& Merging

Repeated for A(x) & B(x)

C(x)

Schoolbook 16 x 16: Pointwise Multiplication

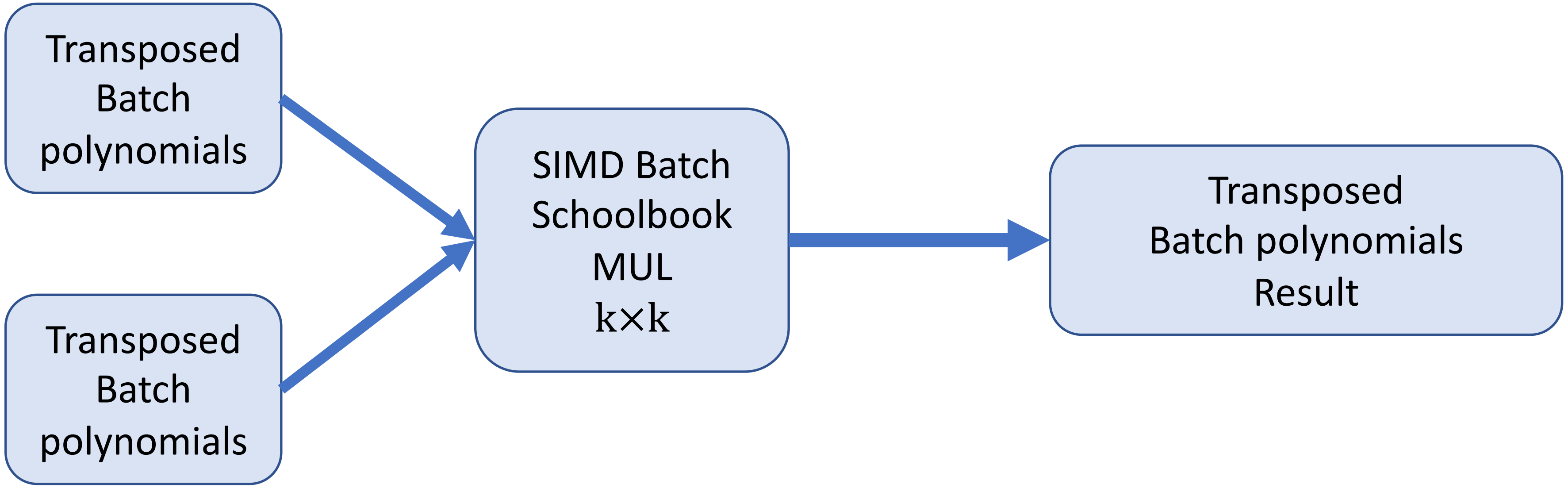
Toom-Cook Implementation: NTRU and Saber



For multiple layers of Split-Evaluate/Interpolate-Merge:

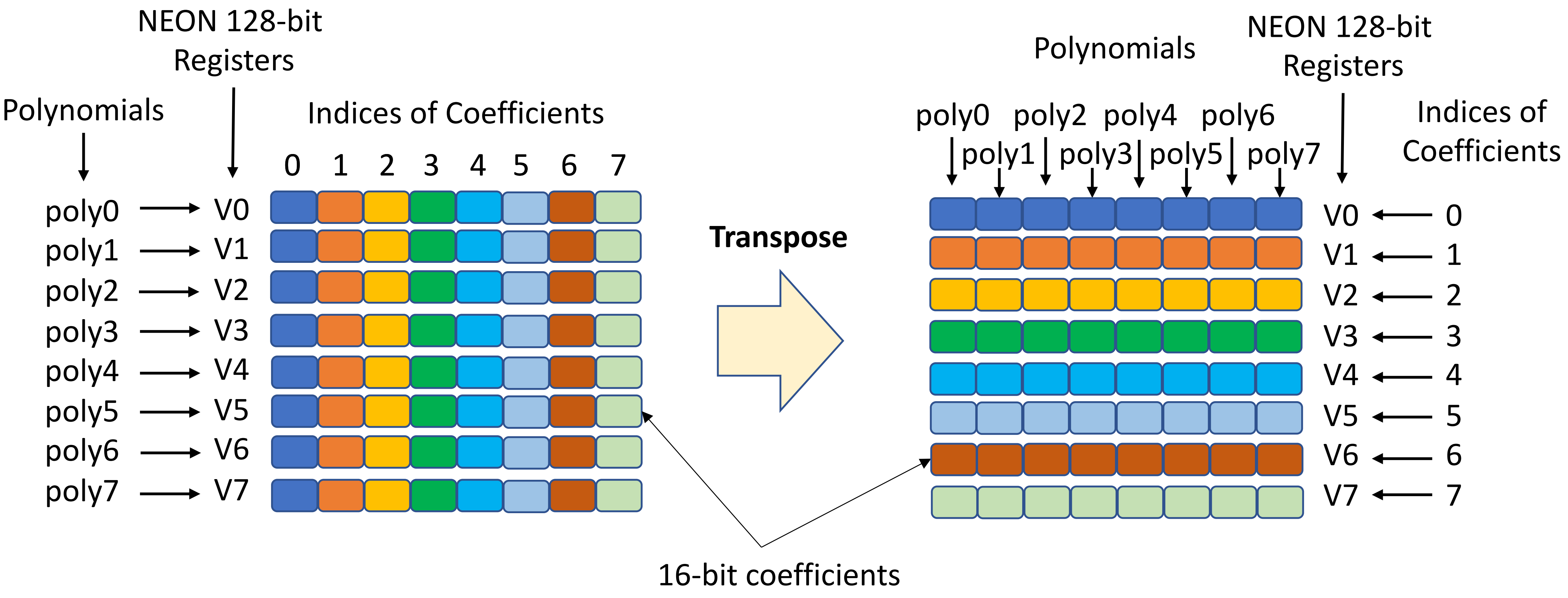
- unroll these layers to save load/store instructions

SIMD Batch Schoolbook Multiplication



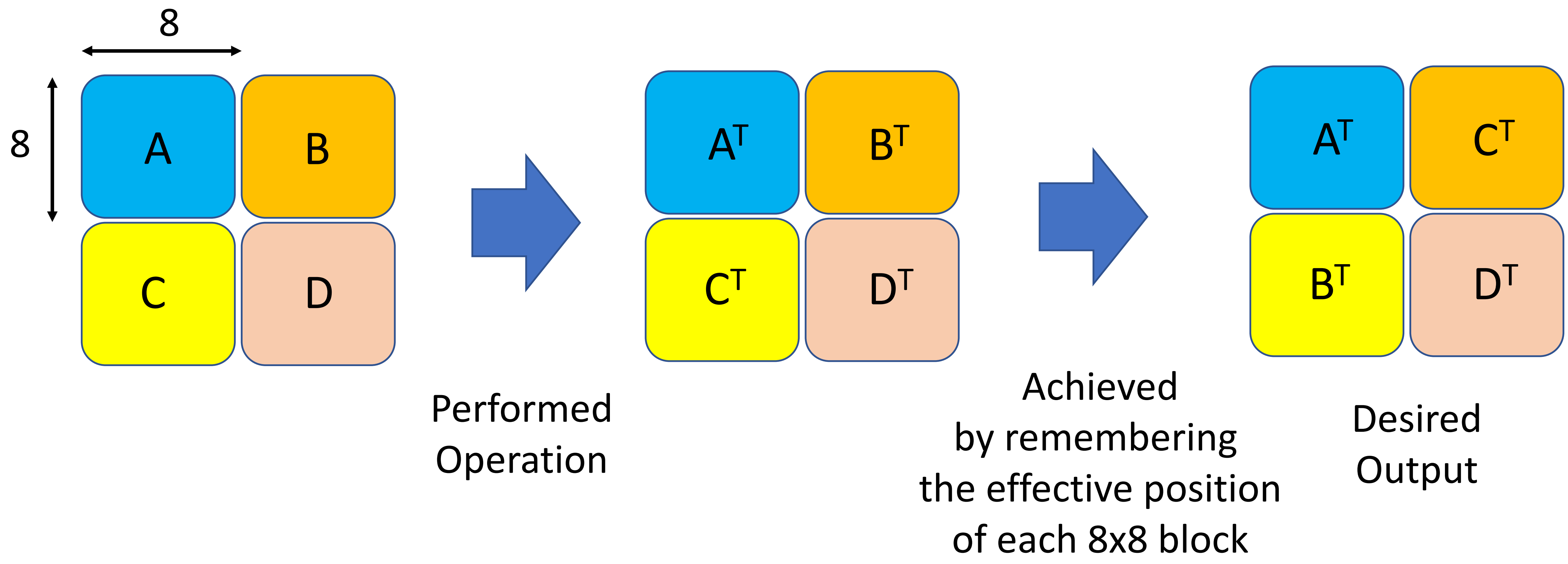
- In order to perform a <u>batch multiplication,</u> a matrix of k polynomials with k coefficients has to be transposed before and after the multiplication
- Optimal value of k was determined to be 16

The 8x8 Matrix Transpose Operation



- The transpose operation enables performing the same operation on the same coefficients of 8 polynomials in parallel
- The 8 x 8 matrix transpose requires 27 out of 32 NEON 128-bit registers

The 16x16 Matrix Transpose Operation



- 16 x 16 matrix transpose requires memory
- To transpose 16 x 16 efficiently, transpose only 8 x 8 matrices and remember the location of each 8 x 8 block

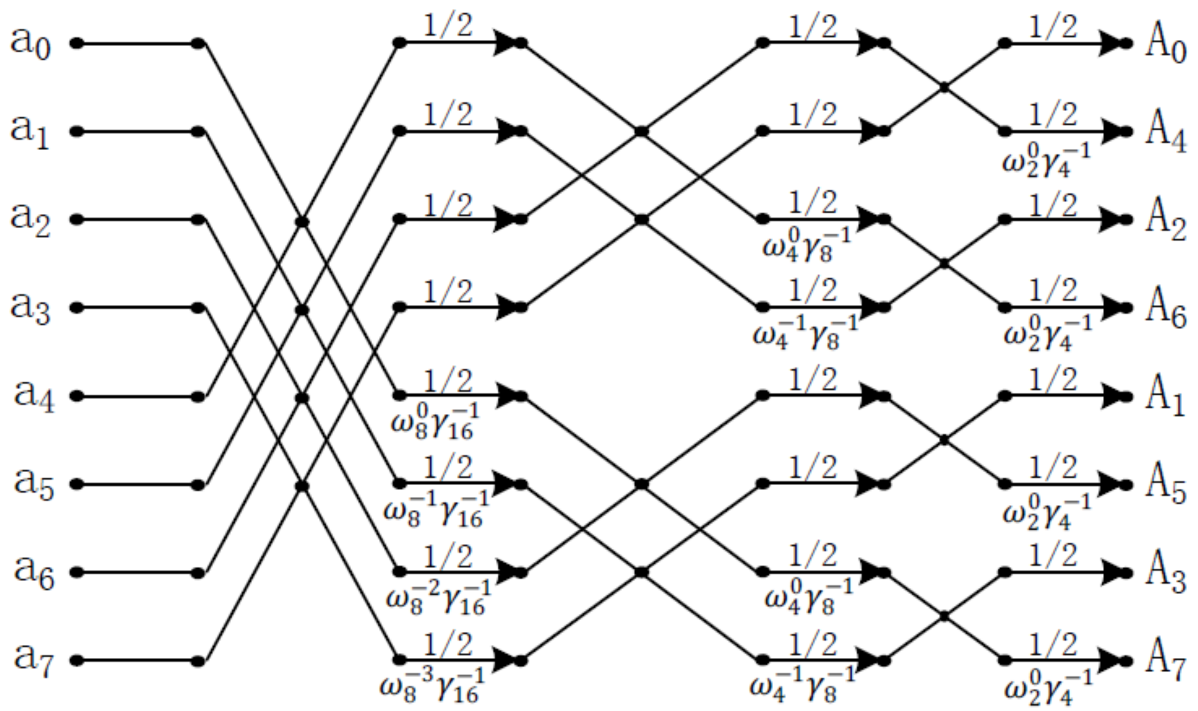
Number Theoretic Transform

Complete NTT

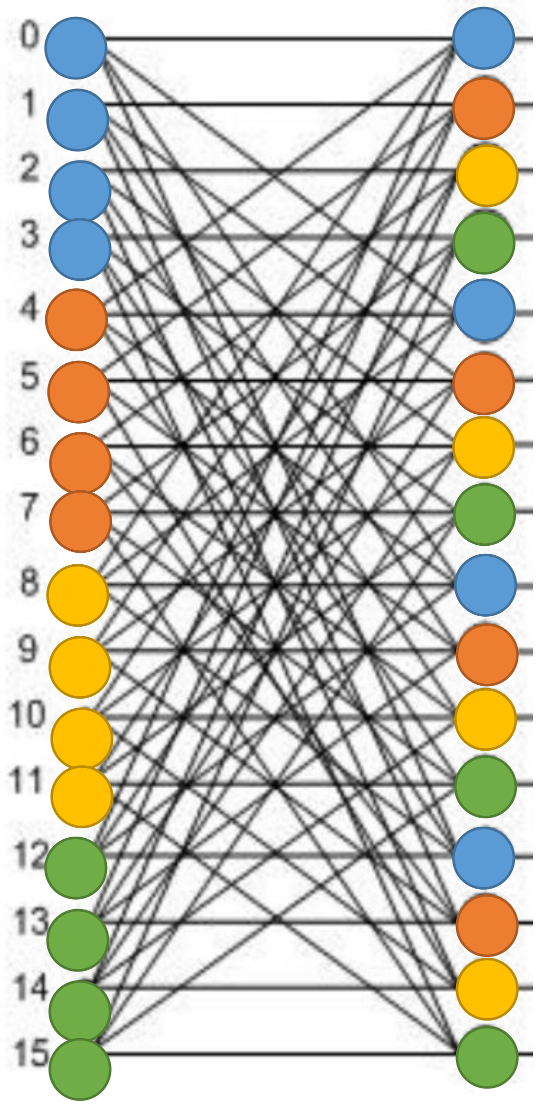
$$
\begin{aligned}
C(x) &= A(x) \times B(x) \\
&= \mathcal{NTT}^{-1}(\mathcal{C}) = \mathcal{NTT}^{-1}(\mathcal{A} * \mathcal{B}) = \mathcal{NTT}^{-1}(\mathcal{NTT}(A) * \mathcal{NTT}(B))
\end{aligned}
$$

where $A(x), B(x), C(x) \epsilon\ Z_q[x]/(x^n + 1)$ $and\ q \equiv 1\ mod\ 2n$

Number Theoretic Transform

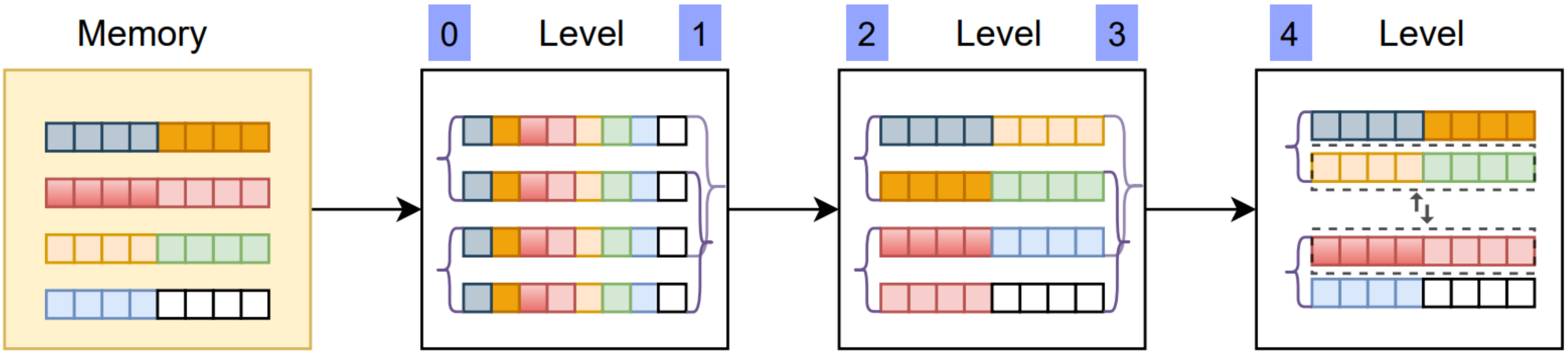


Example of levels



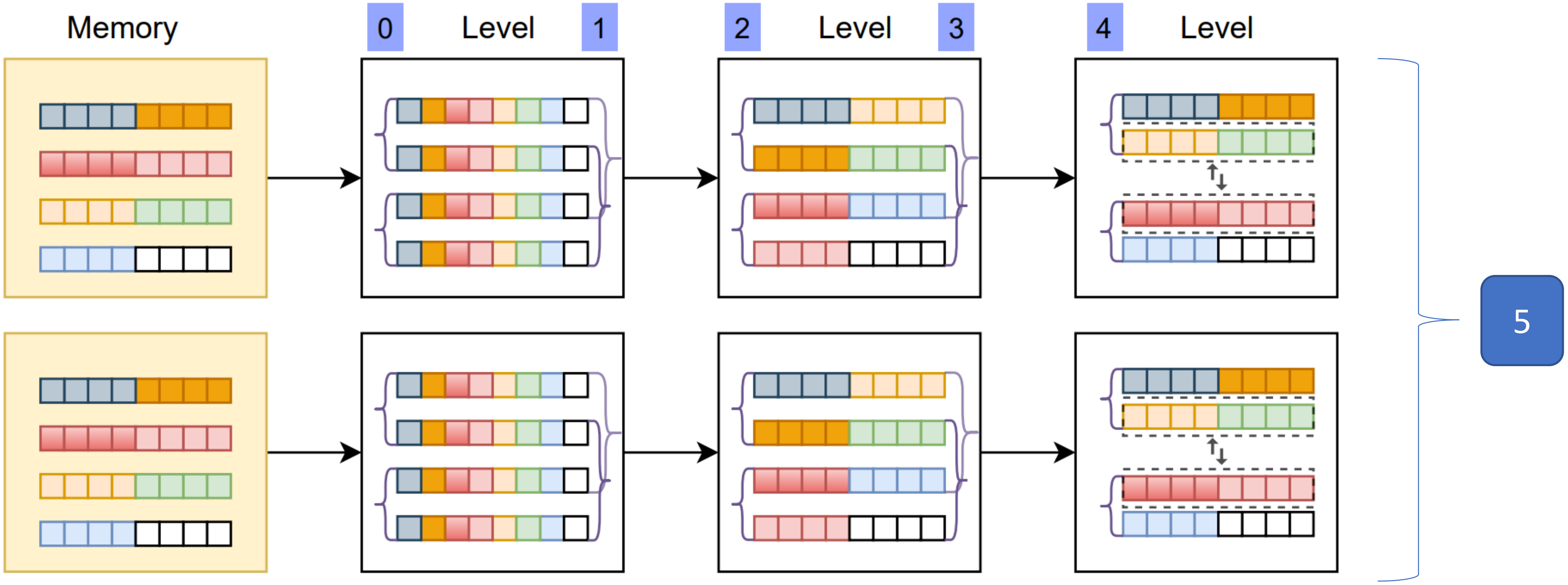
Example of reordering indices between levels

NTT Implementation: CRYSTALS-Kyber and Saber



- Utilize Load and Interleave instructions for Level 0-1
- Use transpose instructions for Level 2-3
- Twist store registers in Level 4

NTT Implementation: CRYSTALS-Kyber and Saber



NTT Implementation: CRYSTALS-Kyber and Saber

- Apply to NTT/FFT based submissions

- 16-bit coefficients can reach level 6

- 32-bit coefficients can reach level 5.

NTT Implementation: Multiply and Modular Reduction

Algorithm 2: Vectorized multiplication modulo a 16-bit q

Input: $B = (B_L, B_H), C = (C_L, C_H), R = 2^{16}$

Output: $A = B * (CR)$ `mod q`

$T_0 \leftarrow \texttt{smull\_s16}(B_L, C_L)$

$T_1 \leftarrow \texttt{smull\_s16}(B_H, C_H)$

$T_2 \leftarrow \texttt{uzp1\_s16}(T_0, T_1)$

$T_3 \leftarrow \texttt{uzp2\_s16}(T_0, T_1)$

$(A_L, A_H) \leftarrow \texttt{mul\_s16}(T_2, q^{-1})$

$T_1 \leftarrow \texttt{smull\_s16}(A_L, q)$

$T_2 \leftarrow \texttt{smull\_s16}(A_H, q)$

$T_0 \leftarrow \texttt{uzp2\_s16}(T_1, T_2)$

$A \leftarrow T_3 - T_0$

Algorithm 15 Multiplication modulo 16-bit q

Require: $-2^{15} \leq a < 2^{15}$, $\frac{q-1}{2} \leq b \leq \frac{q-1}{2}$, $b' = bq^{-1} \bmod 2^{16}$

Ensure: $r \equiv 2^{16}ab \pmod q$

$t_1 \leftarrow \left\lfloor \frac{ab}{2^{16}} \right\rfloor$ ← vpmulhw

$t_0 \leftarrow ab' \bmod 2^{16}$ ← vpmullw

$t_0 \leftarrow \left\lfloor \frac{t_0 q}{2^{16}} \right\rfloor$ ← vpmulhw

$r \leftarrow (t_1 - t_0) \bmod 2^{16}$

- NEON dependency chain: vuzp and vmull (vector unzip and vector multiply)
- Lack of an instruction similar to AVX2 vpmulhw:
  Multiply Packed Unsigned Word Integers and Store the high 16-bits of Result
- Compared to AVX2, our implementation uses additionally
  2 MUL and 3 UNZIP instructions

Benchmarking Methodology

Apple M1 System on Chip	Firestorm core, 3.2 GHz[1], MacBook Air
Broadcom BCM2711 System on Chip	**Cortex-A72** core, 1.5 GHz, Raspberry Pi 4
Operating System	MacOS 11.4, Arch Linux (March 25, 2021)
Compiler	clang 12.0 (MacBook Air), clang 11.1 (Raspberry Pi 4)
Compiler Options	-O3 -mtune=native -fomit-frame-pointer
Cycles count on Cortex-A72	PAPI[2]
Cycles count on Apple M1	Modified[3] from Dougall Johnson's work[4]
Iterations	10,000,000 on Apple M1 to force CPU to run on high-performance "FireStorm" core; 1,000,000 otherwise

1 https://www.anandtech.com/show/16252/mac-mini-apple-m1-tested

2 D. Terpstra, H. Jagode, H. You, and J. Dongarra, "Collecting Performance Data with PAPI-C," in Tools for High Performance Computing, 2009

3 https://github.com/GMUCERG/PQC_NEON/blob/main/neon/kyber/m1cycles.c

4 https://github.com/dougallj

Toom-Cook vs. NTT for Saber

All values in cycles

Apple M1 3.2 Ghz	Encap			Decap		
	Toom	NTT	Toom/NTT	Toom	NTT	Toom/NTT
lightsaber	37,187	43,565	85%	35,318	44,631	79%
saber	59,838	68,867	87%	57,955	71,110	82%
firesaber	87,899	102,206	86%	86,724	106,553	81%

Cortex-A72 1.5 Ghz	Encap			Decap		
	Toom	NTT	Toom/NTT	Toom	NTT	Toom/NTT
lightsaber	131,318	130,097	101%	136,827	131,187	104%
saber	210,383	213,574	99%	218,667	215,364	102%
firesaber	307,360	321,637	96%	323,870	329,566	98%

Dependencies degrade performance of NTT on high-performance processors.

On Apple M1, Toom-Cook better by 13-21%

On Cortex-A72 a tie.

We select Toom-Cook for the implementation of Saber.

Ranking baseline C implementations

Rank	C Cortex-A72 E	kc	↑	D	kc	↑
1	lightsaber	154.8	1.00	lightsaber	165.9	1.00
2	kyber512	184.5	1.19	kyber512	223.4	1.35
3	ntru-hrss701	458.7	2.96	ntru-hps677	1,346.7	8.12
4	ntru-hps677	570.8	3.69	ntru-hrss701	1,353.4	8.16
1	saber	273.4	1.00	saber	294.5	1.00
2	kyber768	298.9	1.09	kyber768	349.1	1.19
3	ntru-hps821	748.1	2.74	ntru-hps821	1,830.0	6.21
1	firesaber	427.3	1.00	firesaber	460.9	1.00
2	kyber1024	440.6	1.03	kyber1024	503.8	1.09

Rank	C Apple M1 E	kc	↑	D	kc	↑
1	lightsaber	50.9	1.00	lightsaber	54.9	1.00
2	kyber512	75.7	1.49	kyber512	89.5	1.63
3	ntru-hrss701	152.4	3.00	ntru-hps677	430.4	7.84
4	ntru-hps677	183.1	3.60	ntru-hrss701	439.9	8.01
1	saber	90.4	1.00	saber	96.2	1.00
2	kyber768	119.8	1.32	kyber768	137.8	1.43
3	ntru-hps821	245.3	2.71	ntru-hps821	586.5	6.10
1	firesaber	140.9	1.00	firesaber	150.8	1.00
2	kyber1024	175.4	1.24	kyber1024	198.4	1.31

Encapsulation and **D**ecapsulation ranking of baseline C implementations:

1. **Saber**
2. **CRYSTALS-Kyber**
3. **NTRU (Levels 1 & 3 only)**

Consistent between Cortex-A72 and Apple M1.

Ranking: NEON implementation

Rank	neon Cortex-A72 E	kc	↑	D	kc	↑
1	ntru-hrss701	93.6	1.00	kyber512	94.1	1.00
2	kyber512	95.3	1.02	lightsaber	131.2	1.39
3	lightsaber	130.1	1.39	ntru-hps677	205.8	2.19
4	ntru-hps677	181.7	1.94	ntru-hrss701	262.9	2.79
1	kyber768	151.0	1.00	kyber768	149.8	1.00
2	saber	213.6	1.41	saber	215.4	1.44
3	ntru-hps821	232.6	1.54	ntru-hps821	274.5	1.83
1	kyber1024	223.8	1.00	kyber1024	220.7	1.00
2	firesaber	321.6	1.44	firesaber	329.6	1.49

Rank	neon Apple M1 E	kc	↑	D	kc	↑
1	ntru-hrss701	22.7	1.00	kyber512	29.4	1.00
2	kyber512	32.5	1.43	lightsaber	35.3	1.20
3	lightsaber	37.2	1.63	ntru-hps677	54.5	1.85
4	ntru-hps677	60.1	2.64	ntru-hrss701	60.7	2.06
1	kyber768	49.2	1.00	kyber768	45.7	1.00
2	saber	59.9	1.22	saber	58.0	1.27
3	ntru-hps821	75.7	1.54	ntru-hps821	69.0	1.51
1	kyber1024	71.6	1.00	kyber1024	67.1	1.00
2	firesaber	87.9	1.23	firesaber	86.7	1.29

Decapsulation ranking of NEON implementations at Levels 1, 3 and 5

Encapsulation ranking of NEON implementations at Level 3 and 5:

1. **CRYSTALS-Kyber**
2. **Saber**
3. **NTRU (Levels 1 & 3 only)**

Consistent between Cortex-A72 and Apple M1.

Exception: Encapsulation at Level 1

1. **NTRU**
2. **CRYSTALS-Kyber**
3. **Saber**

Ranking: C versus NEON

Rank	C Cortex-A72					
	E	*kc*	↑	**D**	*kc*	↑
1	lightsaber	154.8	1.00	lightsaber	165.9	1.00
2	kyber512	184.5	1.19	kyber512	223.4	1.35
3	ntru-hrss701	458.7	2.96	ntru-hps677	1,346.7	8.12
4	ntru-hps677	570.8	3.69	ntru-hrss701	1,353.4	8.16
1	saber	273.4	1.00	saber	294.5	1.00
2	kyber768	298.9	1.09	kyber768	349.1	1.19
3	ntru-hps821	748.1	2.74	ntru-hps821	1,830.0	6.21
1	firesaber	427.3	1.00	firesaber	460.9	1.00
2	kyber1024	440.6	1.03	kyber1024	503.8	1.09

Rank	neon Cortex-A72					
	E	*kc*	↑	**D**	*kc*	↑
1	ntru-hrss701	93.6	1.00	kyber512	94.1	1.00
2	kyber512	95.3	1.02	lightsaber	131.2	1.39
3	lightsaber	130.1	1.39	ntru-hps677	205.8	2.19
4	ntru-hps677	181.7	1.94	ntru-hrss701	262.9	2.79
1	kyber768	151.0	1.00	kyber768	149.8	1.00
2	saber	213.6	1.41	saber	215.4	1.44
3	ntru-hps821	232.6	1.54	ntru-hps821	274.5	1.83
1	kyber1024	223.8	1.00	kyber1024	220.7	1.00
2	firesaber	321.6	1.44	firesaber	329.6	1.49

Why do the rankings of Saber and CRYSTALS-Kyber switch places between the baseline C and NEON implementations?

Answer: **Performance of polynomial multiplication in vector by vector and matrix by vector multiplications**

Saber vs. Kyber

Cortex-A72 1500 MHz	Level 1 *(kilocycles)* ref	 neon	 ref/neon	Level 3 *(kilocycles)* ref	 neon	 ref/neon
Saber: Toom-Cook | NTT						
InnerProd	27.7	18.1 | 22.5	1.53 | 1.23	41.4	25.0 | 31.5	1.64 | 1.31
MatrixVectorMul	55.2	40.2 | 37.0	1.37 | 1.49	125.7	81.0 | 71.3	1.55 | 1.76
Kyber						
VectorVectorMul	44.4	7.1	6.3	59.7	9.9	6.1
MatrixVectorMul	68.1	10.7	6.4	117.5	19.3	6.1

NEON vs. Baseline Speed-Up

Algorithm	ref (kc)		neon (kc)		ref/neon	
	E	**D**	**E**	**D**	**E**	**D**
lightsaber	50.9	54.9	37.2	35.3	1.37	1.55
kyber512	75.7	89.5	32.6	29.4	2.33	3.04
ntru-hps677	183.1	430.4	60.1	54.6	3.05	7.89
ntru-hrss701	152.4	439.9	22.8	60.8	6.68	7.24
saber	90.4	96.2	59.9	58.0	1.51	1.66
kyber768	119.8	137.8	49.2	45.7	2.43	3.02
ntru-hps821	245.3	586.5	75.7	69.0	3.24	8.49
firesaber	140.9	150.8	87.9	86.7	1.60	1.74
kyber1024	175.4	198.4	71.6	67.1	2.45	2.96

Ranking: AVX2

Rank	avx2 AMD EPYC 7742					
	E	kc	↑	**D**	kc	↑
1	ntru-hrss701	20.4	1.00	kyber512	22.5	1.00
2	ntru-hps677	25.9	1.27	lightsaber	42.1	1.87
3	kyber512	28.3	1.39	ntru-hps677	45.7	2.03
4	lightsaber	41.9	2.05	ntru-hrss701	47.6	2.11
1	ntru-hps821	29.9	1.00	kyber768	35.2	1.00
2	kyber768	43.4	1.45	saber	57.3	1.63
3	saber	70.9	2.37	ntru-hps821	70.7	2.01
1	kyber1024	63.0	1.00	kyber1024	53.1	1.00
2	firesaber	103.3	1.64	firesaber	103.7	1.95

For **D**ecapsulation, the rankings across all security levels are:

1. CRYSTALS-Kyber, 2.Saber, 3. NTRU (Levels 1 & 3 only)

For **E**ncapsulation, at levels 1 and 3, the rankings are:

1. NTRU, 2. CRYSTALS-Kyber, 3. Saber

For **E**ncapsulation, at level 5, the ranking is:

1. CRYSTALS-Kyber, 2. Saber

NEON vs. AVX2 in Cycles

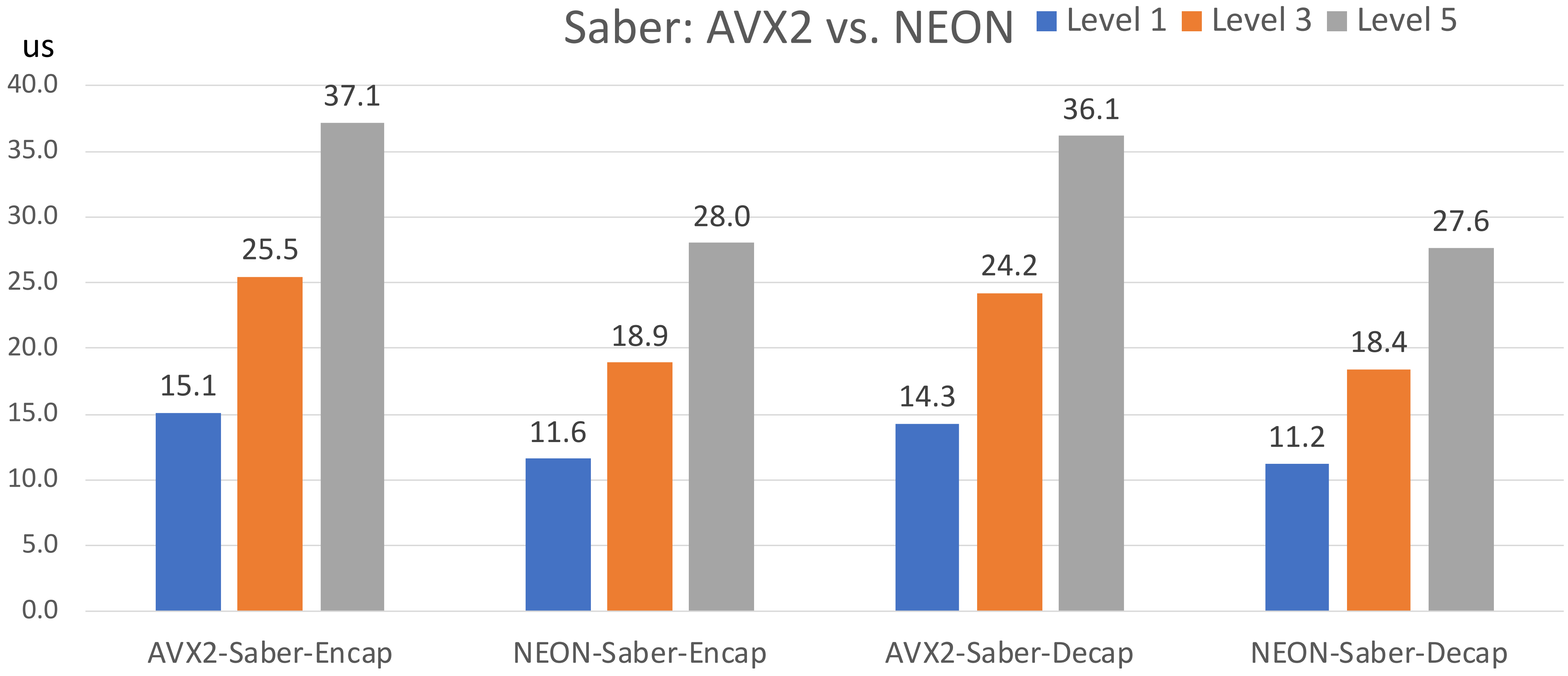
Algorithm	neon (kc)		AVX2 (kc)		AVX2/neon	
	E	D	E	D	E	D
lightsaber	37.2	35.3	41.9	42.2	1.13	1.19
kyber512	32.6	29.4	28.4	22.6	0.87	0.77
ntru-hps677	60.1	54.6	26.0	45.7	0.43	0.84
ntru-hrss701	22.8	60.8	20.4	47.7	0.90	0.78
saber	59.9	58.0	70.9	70.7	1.18	1.22
kyber768	49.2	45.7	43.4	35.2	0.88	0.77
ntru-hps821	75.7	69.0	29.9	57.3	0.39	0.83
firesaber	87.9	86.7	103.3	103.7	1.18	1.20
kyber1024	71.6	67.1	63.0	53.1	0.88	0.79

Result for AVX2 AMD EPYC 7742 taken from **supercop-20210125**

Frequency Scaling Effect
Apple M1 @ **3.2 GHz** versus Intel Core i7-8750H **4.1 GHz**

Apple M1	ref (kc)		neon (kc)		AVX2 (kc)		ref/neon		AVX2/neon	
Core i7-8750H	E	D	E	D	E	D	E	D	E	D
NTRU-HPS677	183.1	430.4	60.1	54.6	47.6	32.5	3.05	7.89	0.79	0.60
NTRU-HRSS701	152.4	439.9	22.8	60.8	28.8	33.9	6.68	7.24	1.26	0.56
LIGHTSABER	50.9	54.9	37.2	35.3	35.1	32.3	1.37	1.55	0.94	0.91
KYBER512	75.7	89.5	32.6	29.4	23.2	17.5	2.33	3.04	0.71	0.59
NTRU-HPS821	245.3	586.5	75.7	69.0	56.1	40.7	3.24	8.49	0.74	0.59
SABER	90.4	96.2	59.9	58.0	54.3	53.8	1.51	1.66	0.91	0.93
KYBER768	119.8	137.8	49.2	45.7	33.9	26.0	2.43	3.02	0.69	0.57
FIRESABER	140.9	150.8	87.9	86.7	78.9	78.1	1.60	1.74	0.90	0.90
KYBER1024	175.4	198.4	71.6	67.1	45.2	35.5	2.45	2.96	0.63	0.53

Frequency Scaling Effect
Apple M1 @ **3.2 GHz** versus Intel Core i7-8750H **4.1 GHz**



Time measured with the ns accuracy using clock_gettime() on a MacBook Air and a PC laptop

Conclusions: Toom-Cook and NTT

- The polynomial multiplication performance affects the C baseline and NEON rankings in case of Saber and Kyber.

- Proposed optimal Toom-Cook strategy tailored for NTRU and Saber parameters.

- Missing instruction equivalent to AVX2 vpmulhw causes dependencies and worse performance

Conclusions

- First optimized implementation of CRYSTALS-Kyber, NTRU, and Saber targeting ARMv8.

- Largest speed-up for NTRU, followed by CRYSTALS-Kyber, and Saber

- The rankings of lattice-based PQC KEM finalists in terms of speed in software are similar for the NEON implementations and AVX2 implementations

  Decapsulation: 1. CRYSTALS-Kyber, 2.Saber, 3. NTRU (L1 & 3 only)

  Encapsulation: 1. NTRU (L1 & 3 only), 2. CRYSTALS-Kyber, 3. Saber

Thanks for your attention!

Our source code is available at:
https://github.com/GMUCERG/PQC_NEON