

# Fast NEON-based multiplication for lattice-based NIST Post-Quantum Cryptography finalists

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).



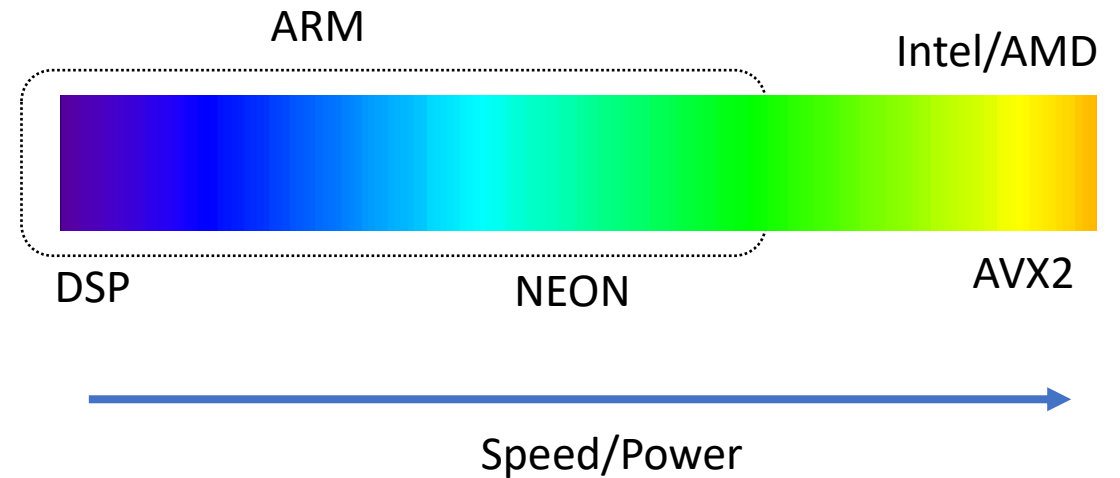
Apple M1:  
part of new MacBook Air, MacBook Pro,  
Mac Mini, iMac, and iPad Pro



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

# Introduction

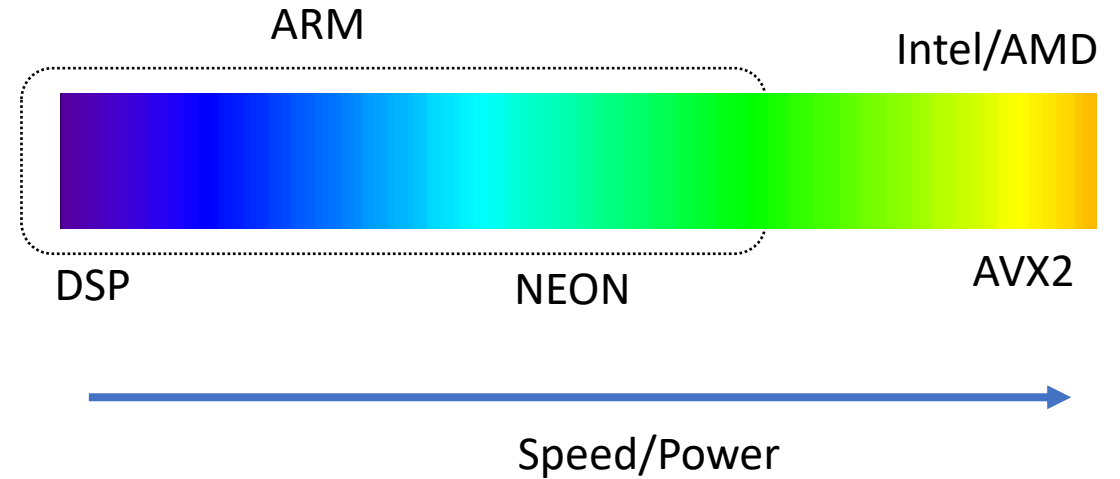
- Most software implementations of PQC candidates on:
  - Intel/AMD (w/ AVX2 extension)
  - Cortex-M4 (w/ DSP extension)<sup>1</sup>
- Lack of NEON implementations on ARMv7 and ARMv8 architectures



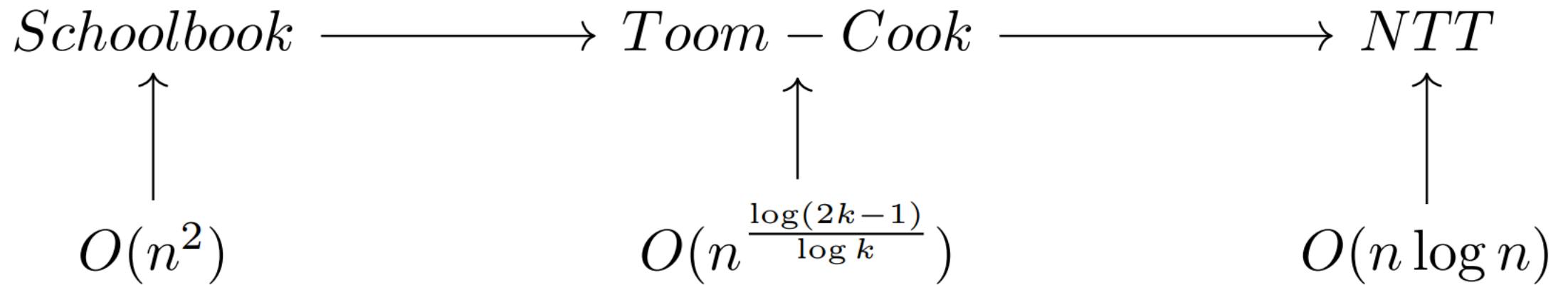
<sup>1</sup> M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen, pqm4 - Post-quantum crypto library for the ARM Cortex-M4, <https://github.com/mupq/pqm4>

# 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



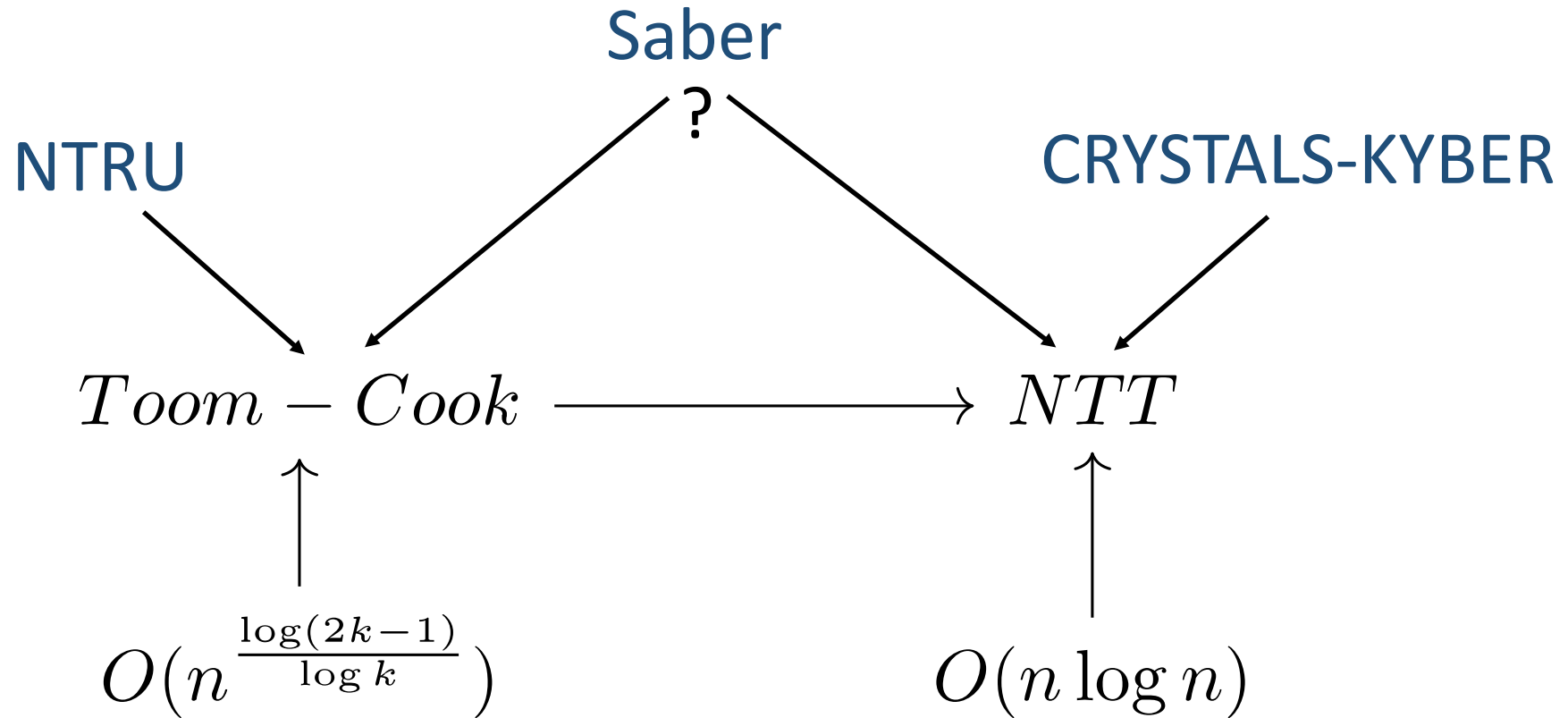
Typically:

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

$$\begin{aligned} A(x) &= x^{\frac{3n}{4}} \sum_{i=\frac{3n}{4}}^{n-1} a_i x^{(i-\frac{3n}{4})} + \dots + x^{\frac{n}{4}} \sum_{i=\frac{n}{4}}^{\frac{2n}{4}-1} a_i x^{(i-\frac{n}{4})} + \sum_{i=0}^{\frac{n}{4}-1} a_i x^i \\ &= \alpha_3 \cdot x^{\frac{3n}{4}} + \alpha_2 \cdot x^{\frac{2n}{4}} + \alpha_1 \cdot x^{\frac{n}{4}} + \alpha_0 \end{aligned}$$

## 2. Evaluation

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

## 3. Pointwise multiplication

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

# 5 Steps of Toom-4

## 4. Interpolation

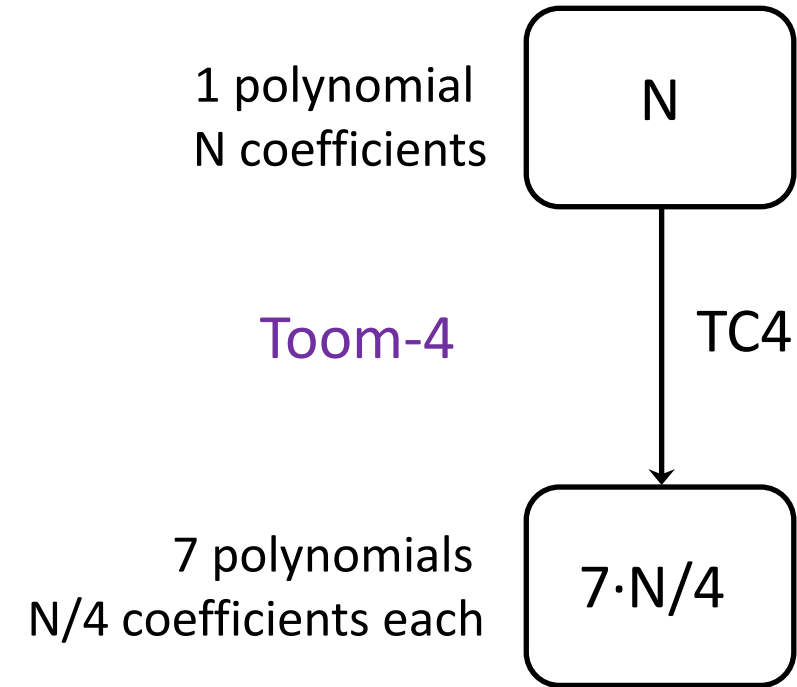
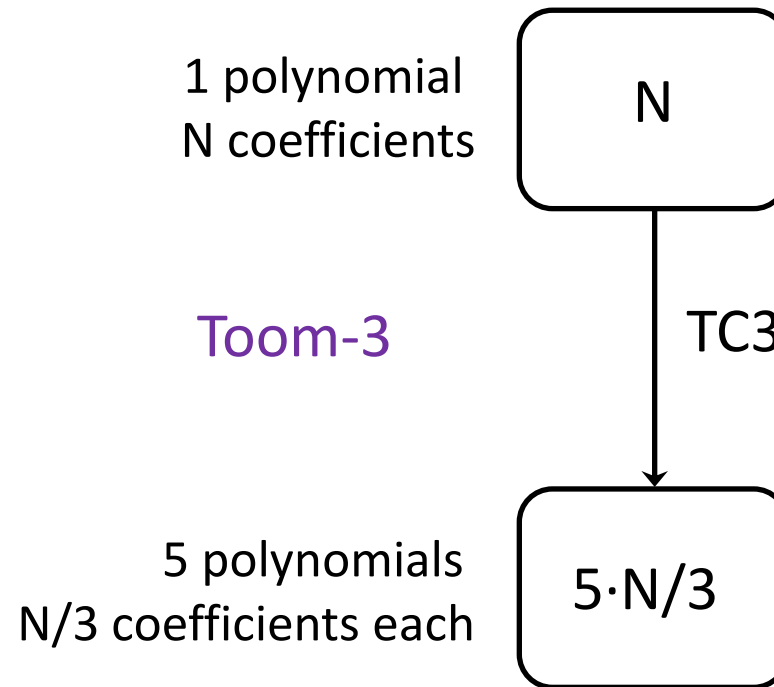
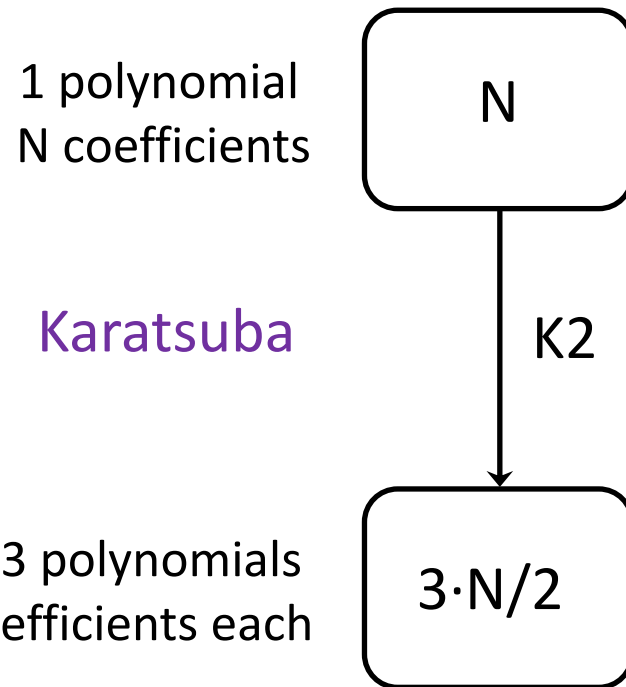
$$\begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ \theta_6 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 \\ \frac{1}{64} & \frac{1}{32} & \frac{1}{16} & \frac{1}{8} & \frac{1}{4} & \frac{1}{2} & 1 \\ \frac{1}{64} & -\frac{1}{32} & \frac{1}{16} & -\frac{1}{8} & \frac{1}{4} & -\frac{1}{2} & 1 \\ 64 & 32 & 16 & 8 & 4 & 2 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^{-1} \cdot \begin{bmatrix} \mathcal{C}(0) \\ \mathcal{C}(1) \\ \mathcal{C}(-1) \\ \mathcal{C}(\frac{1}{2}) \\ \mathcal{C}(-\frac{1}{2}) \\ \mathcal{C}(2) \\ \mathcal{C}(\infty) \end{bmatrix} \quad \text{where} \quad \mathcal{C}(\mathcal{X}) = \sum_{i=0}^6 \theta_i \mathcal{X}^i$$

## 5. Merging

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

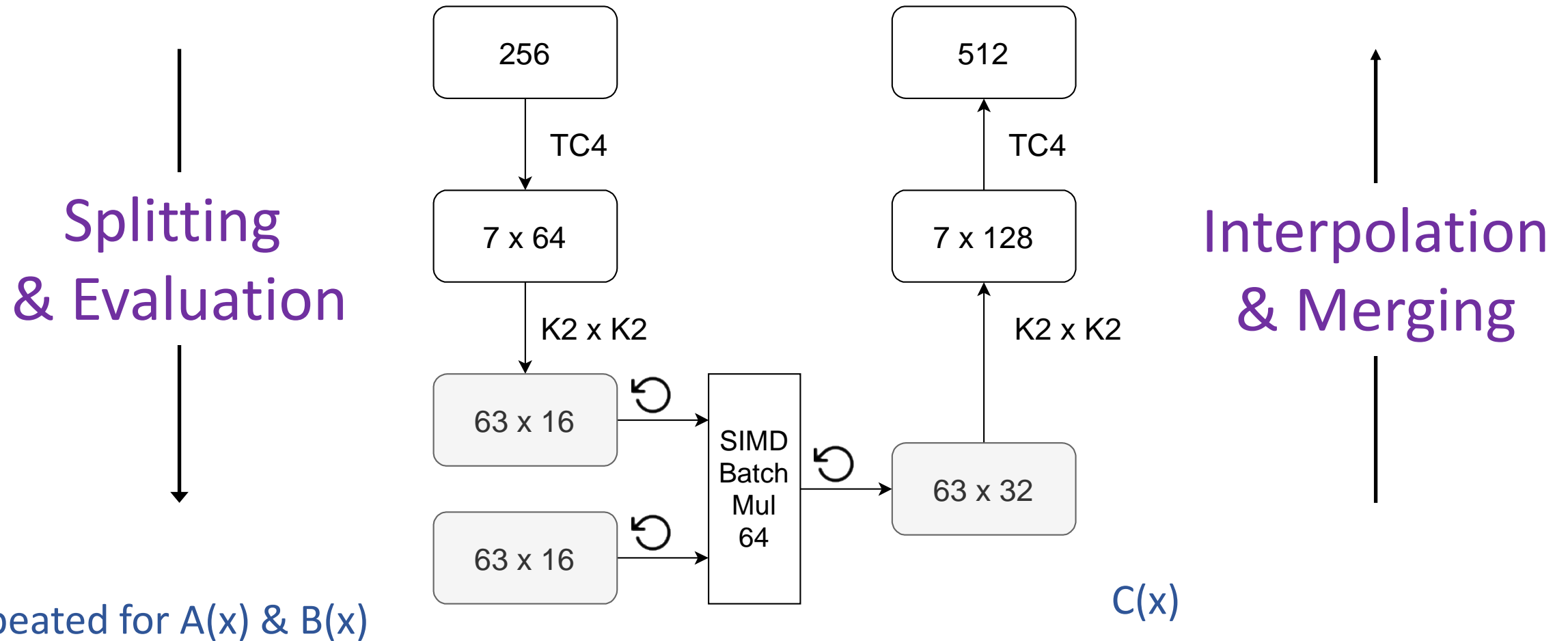


# Toom-Cook: Splitting & Evaluation



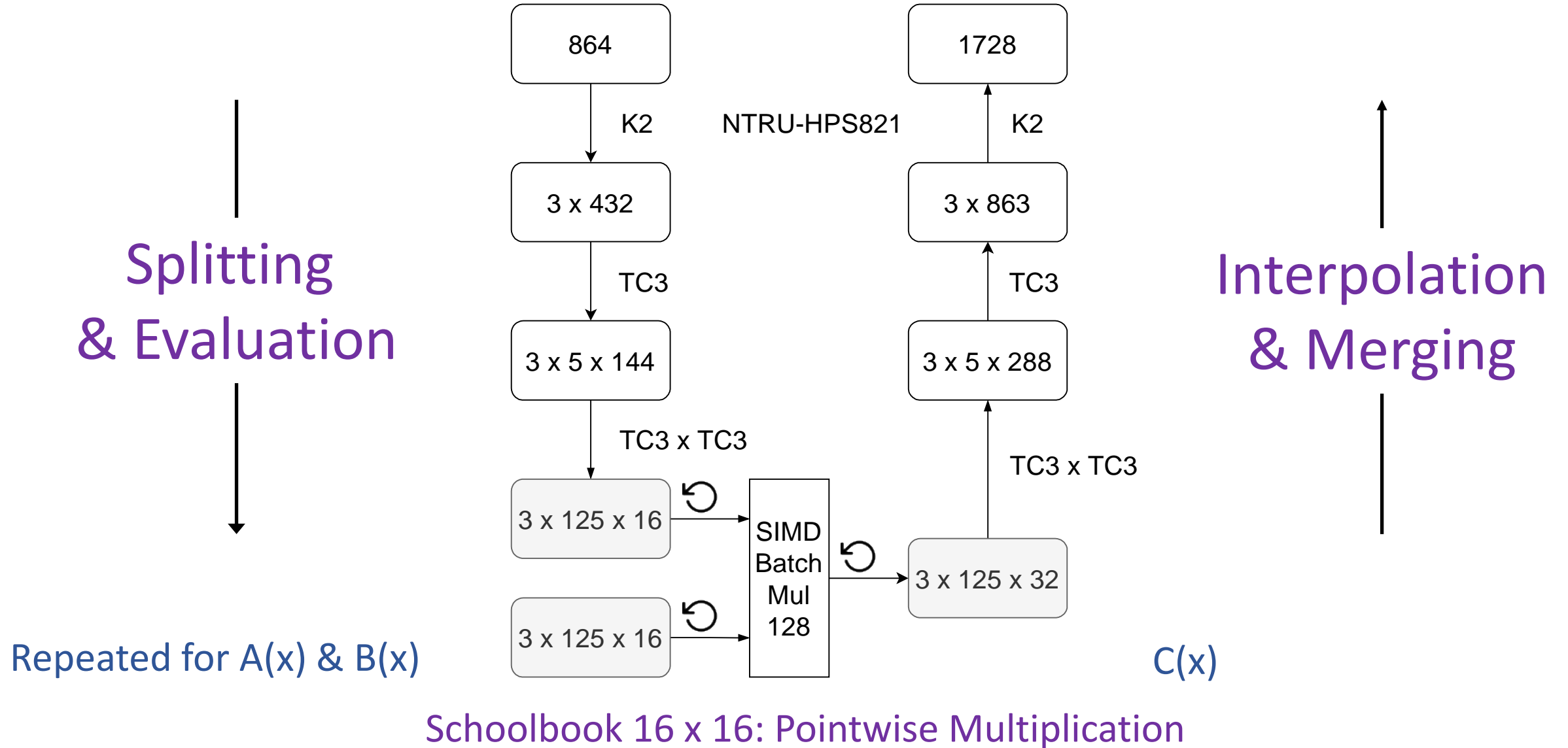
# Toom-Cook Implementation: Saber

SABER

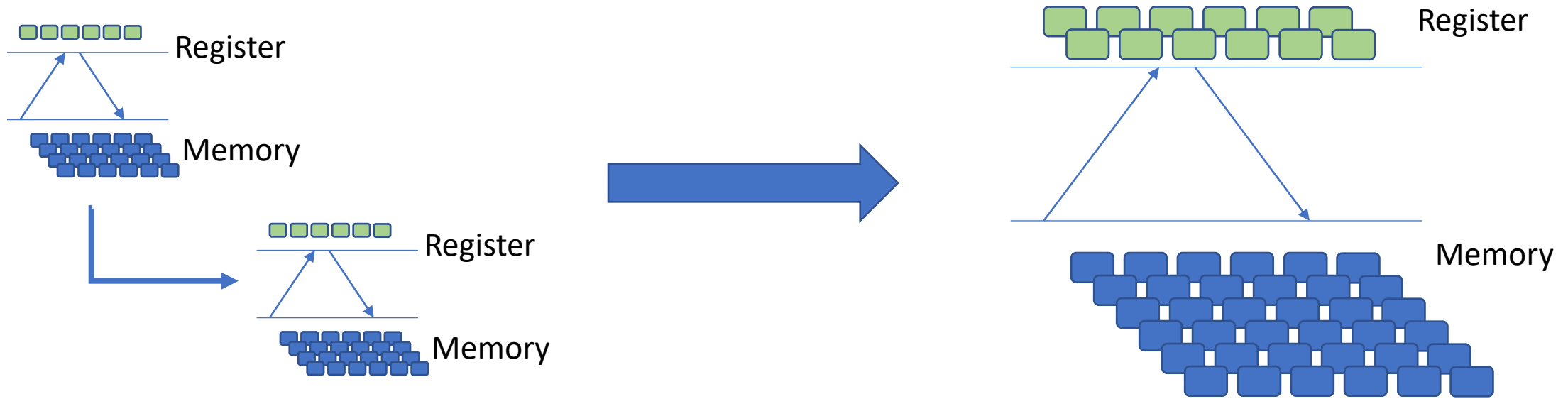


Schoolbook 16 x 16: Pointwise Multiplication

# Toom-Cook Implementation: NTRU-HPS821



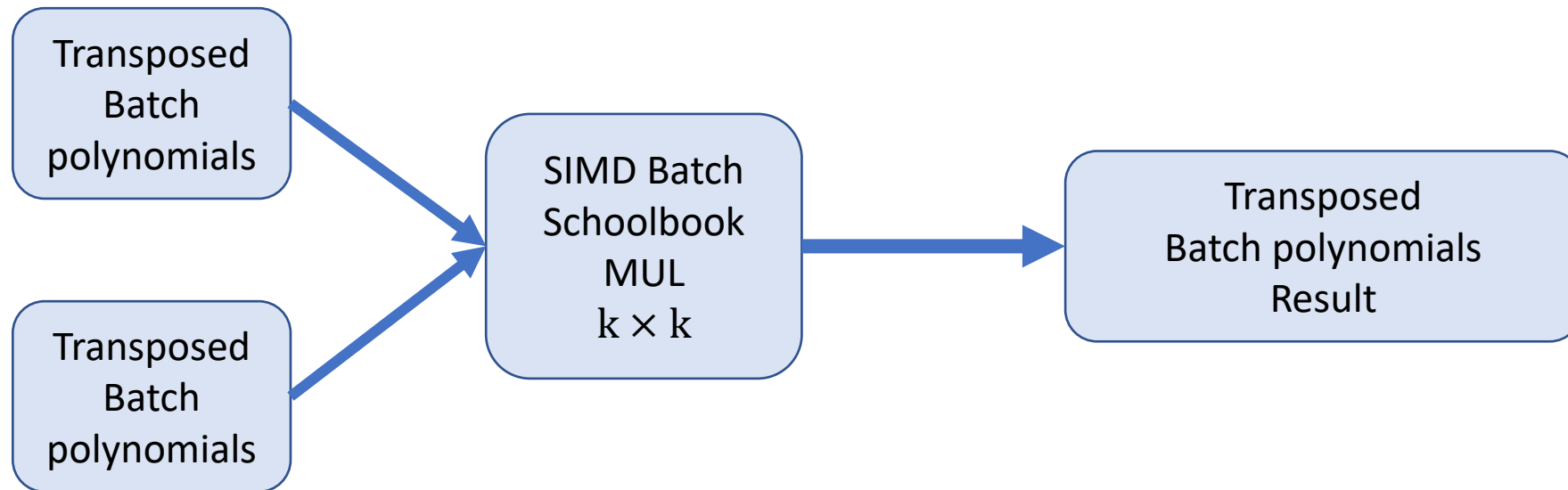
# Toom-Cook Implementation: NTRU and Saber



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

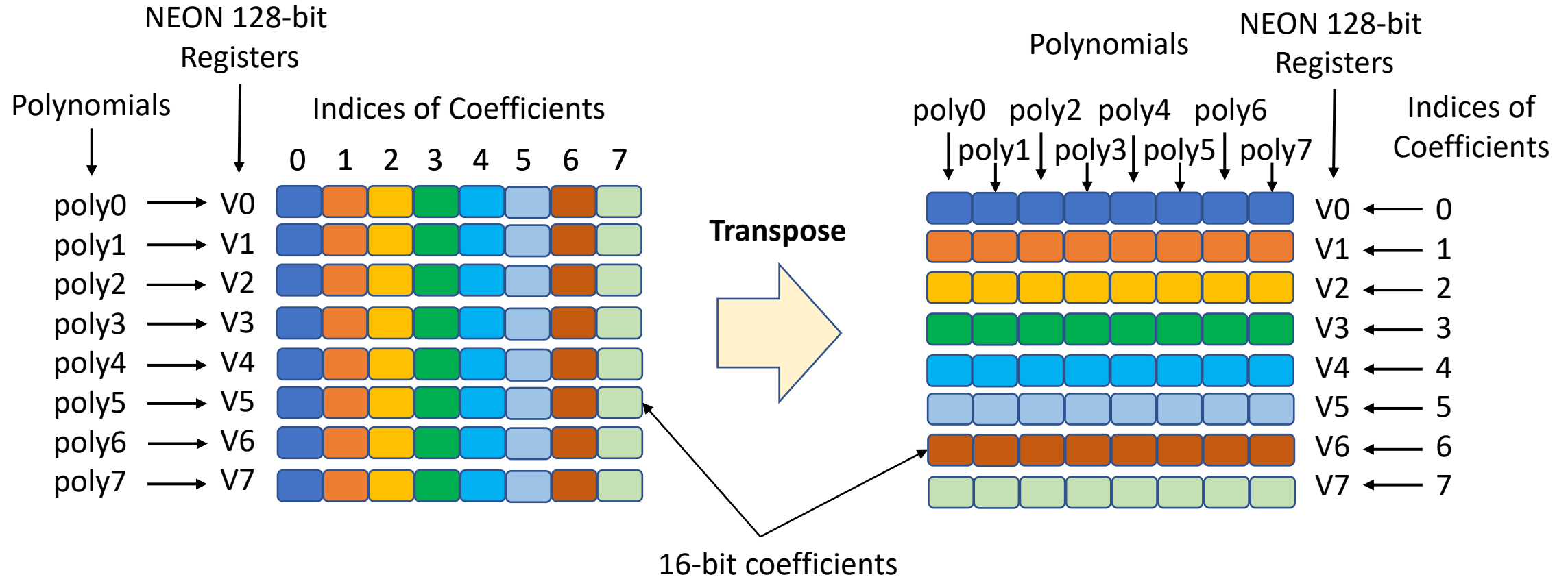
- unroll these layers to save load/store instructions

# SIMD Batch Schoolbook Multiplication



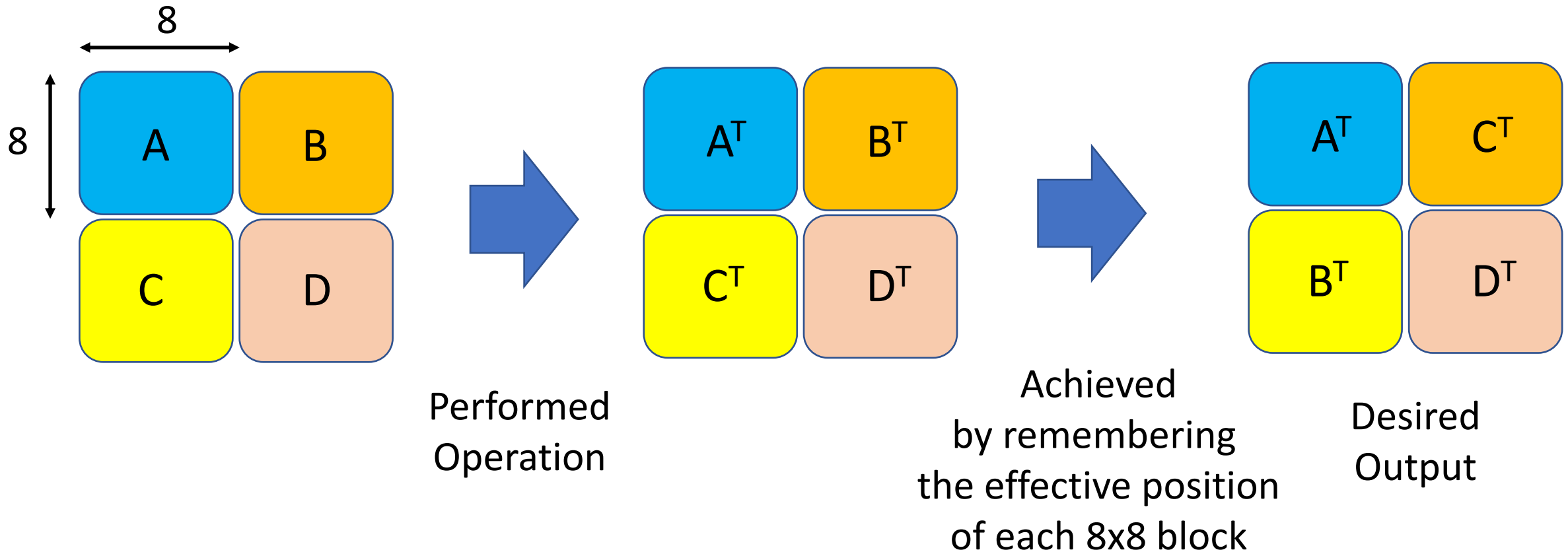
- In order to perform a batch multiplication, a matrix of  $k$  polynomials with  $k$  coefficients has to be transposed before and after the multiplication
- Optimal value of  $k$  was determined to be 16

# 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

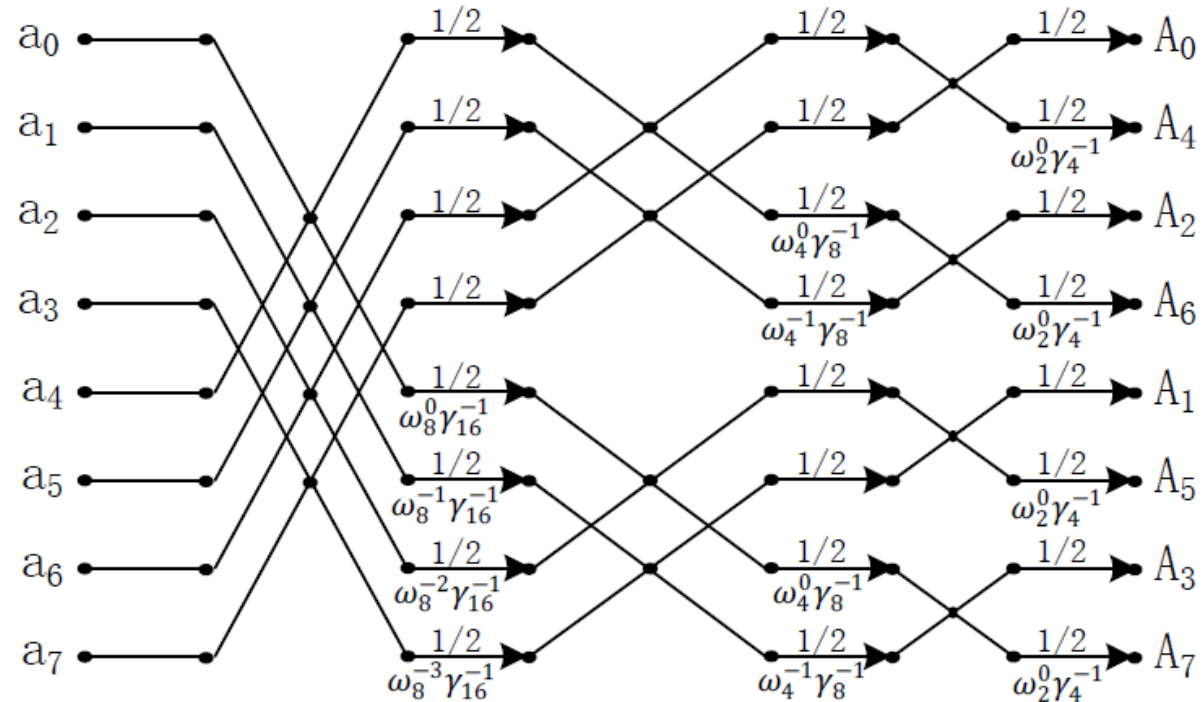
## Complete NTT

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

where  $A(x), B(x), C(x) \in \mathbb{Z}_q[x]/(x^n + 1)$  and  $q \equiv 1 \pmod{2n}$

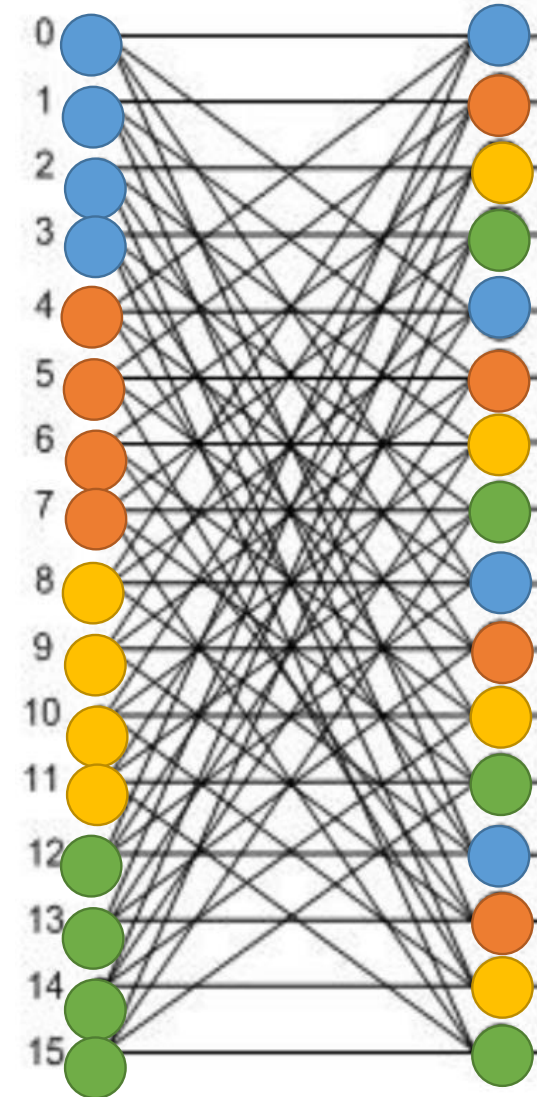


# Number Theoretic Transform

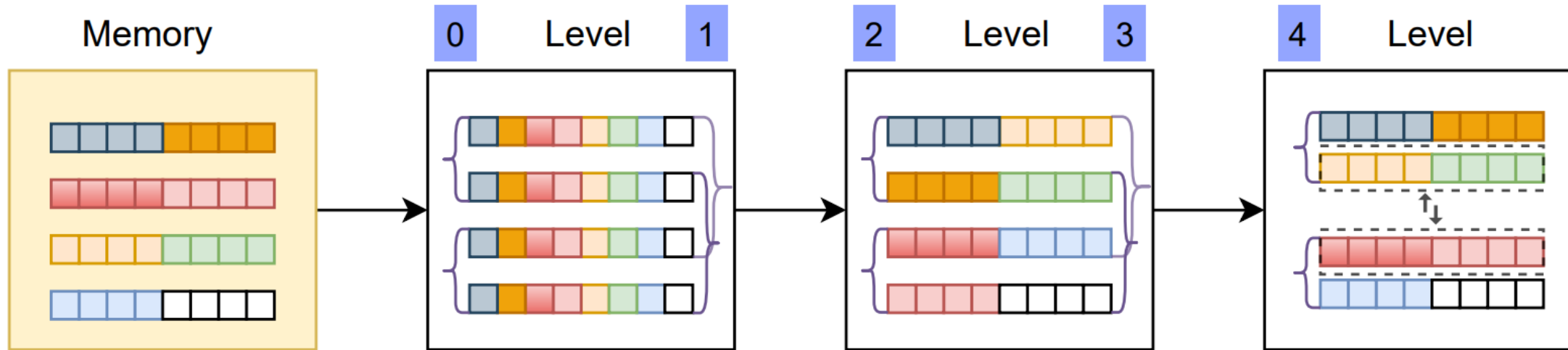


Example of levels

Example of reordering indices  
between levels

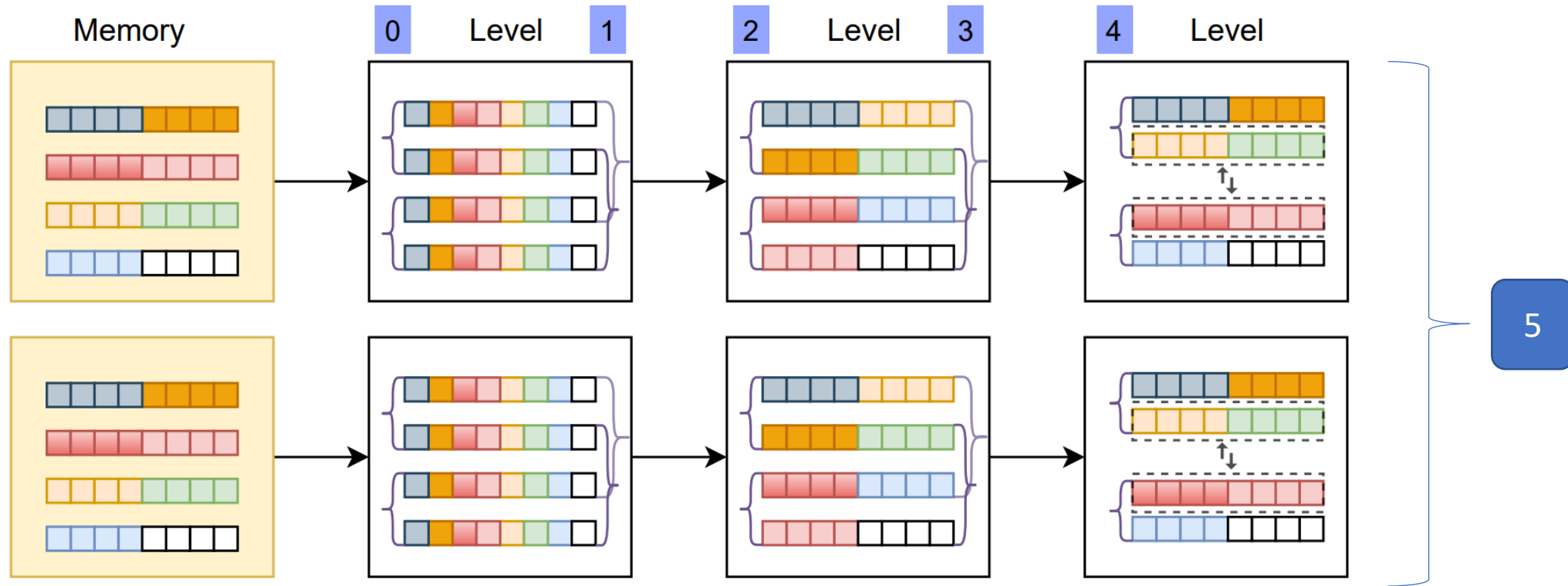


# NTT Implementation: CRYSTALS-Kyber and Saber



- 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:**  $A = B * (CR) \bmod q$

```

1  $T_0 \leftarrow \text{smull\_s16}(B_L, C_L)$ 
2  $T_1 \leftarrow \text{smull\_s16}(B_H, C_H)$ 
3  $T_2 \leftarrow \text{uzp1\_s16}(T_0, T_1)$ 
4  $T_3 \leftarrow \text{uzp2\_s16}(T_0, T_1)$ 
5  $(A_L, A_H) \leftarrow \text{mul\_s16}(T_2, q^{-1})$ 
6  $T_1 \leftarrow \text{smull\_s16}(A_L, q)$ 
7  $T_2 \leftarrow \text{smull\_s16}(A_H, q)$ 
8  $T_0 \leftarrow \text{uzp2\_s16}(T_1, T_2)$ 
9  $A \leftarrow T_3 - T_0$ 

```

---

Chung et. al **NTT Multiplication for NTT-unfriendly Rings**

---

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

```

1:  $t_1 \leftarrow \left\lfloor \frac{ab}{2^{16}} \right\rfloor$  ← vpmulhw
2:  $t_0 \leftarrow ab' \bmod 2^{16}$  ← vpmullw
3:  $t_0 \leftarrow \left\lfloor \frac{t_0 q}{2^{16}} \right\rfloor$  ← vpmulhw
4:  $r \leftarrow (t_1 - t_0) \bmod 2^{16}$ 

```

---

- NEON dependency chain: `uzp` and `smull` (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

# 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.

**Old results in the published PQCrypto paper**

# Improved 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:**  $A = B * (CR) \bmod q$

```
1  $T_0 \leftarrow \text{smull\_s16}(B_L, C_L)$ 
2  $T_1 \leftarrow \text{smull\_s16}(B_H, C_H)$ 
3  $T_2 \leftarrow \text{uzp1\_s16}(T_0, T_1)$ 
4  $T_3 \leftarrow \text{uzp2\_s16}(T_0, T_1)$ 
5  $(A_L, A_H) \leftarrow \text{mul\_s16}(T_2, q^{-1})$ 
6  $T_1 \leftarrow \text{smull\_s16}(A_L, q)$ 
7  $T_2 \leftarrow \text{smull\_s16}(A_H, q)$ 
8  $T_0 \leftarrow \text{uzp2\_s16}(T_1, T_2)$ 
9  $A \leftarrow T_3 - T_0$ 
```

---

Improved NTT implementation

---

**Algorithm 3:** Vectorized multiplication modulo a 16-bit  $q$

---

**Input:**  $B, C, R = 2^{16}, B_{qinv} = B * q^{-1} \bmod 2^{16}$

**Output:**  $A = B * (CR) \bmod q$

```
1  $T_0 \leftarrow \text{sqdmulh\_s16}(B, C)$ 
2  $T_1 \leftarrow \text{mul\_s16}(B_{qinv}, C)$ 
3  $T_2 \leftarrow \text{sqdmulh\_s16}(T_1, q)$ 
4  $A \leftarrow \text{shsub\_s16}(T_2 - T_0)$ 
```

---

- Resolved NEON dependency chain
- Found alternative instruction to AVX2 vpmulhw: **sqdmulh\_s16**  
Signed saturating Doubling Multiply store the high 16-bits of Result
- Compared to AVX2, our **new NTT implementation** uses the **same number of instructions**.



# Improved 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	35,949	103%	35,318	34,142	103%
saber	59,838	55,892	107%	57,955	54,117	107%
firesaber	87,899	82,776	106%	86,724	81,983	106%

Cortex-A72 1.5 Ghz	Encap			Decap		
	Toom	NTT	Toom/NTT	Toom	NTT	Toom/NTT
lightsaber	130,097	116,105	112%	131,187	115,859	113%
saber	213,574	183,230	116%	215,364	183,208	117%
firesaber	321,637	265,626	121%	329,566	270,989	121%

After resolving dependency, the performance of NTT increases.

On Apple M1, NTT better by 3-7%

On Cortex-A72, NTT better by 12-21%

**New results to appear in the ePrint paper**



# New versus Previous NEON Results

- All subsequent NEON results have been obtained using the previous version of the code available at the time of the final paper submission.
- New results will be presented during a short 10-minute talk during the conference and in a follow-up ePrint paper.
- Despite a substantial improvement in absolute values of execution times for Kyber and Saber, all rankings of candidates have remained unchanged.

# Benchmarking Methodology

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

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

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

<sup>3</sup> [https://github.com/GMUCERG/PQC\\_NEON/blob/main/neon/kyber/m1cycles.c](https://github.com/GMUCERG/PQC_NEON/blob/main/neon/kyber/m1cycles.c)

<sup>4</sup> <https://github.com/dougallj>

# Ranking baseline C implementations

Rank	C Cortex-A72						Rank	C Apple M1					
	E	kc	↑	D	kc	↑		E	kc	↑	D	kc	↑
1	lightsaber	154.8	<u>1.00</u>	lightsaber	165.9	<u>1.00</u>	1	lightsaber	50.9	<u>1.00</u>	lightsaber	54.9	<u>1.00</u>
2	kyber512	184.5	1.19	kyber512	223.4	1.35	2	kyber512	75.7	1.49	kyber512	89.5	1.63
3	ntru-hrss701	458.7	2.96	ntru-hps677	1,346.7	8.12	3	ntru-hrss701	152.4	3.00	ntru-hps677	430.4	7.84
4	ntru-hps677	570.8	3.69	ntru-hrss701	1,353.4	8.16	4	ntru-hps677	183.1	3.60	ntru-hrss701	439.9	8.01
1	saber	273.4	<u>1.00</u>	saber	294.5	<u>1.00</u>	1	saber	90.4	<u>1.00</u>	saber	96.2	<u>1.00</u>
2	kyber768	298.9	1.09	kyber768	349.1	1.19	2	kyber768	119.8	1.32	kyber768	137.8	1.43
3	ntru-hps821	748.1	2.74	ntru-hps821	1,830.0	6.21	3	ntru-hps821	245.3	2.71	ntru-hps821	586.5	6.10
1	firesaber	427.3	<u>1.00</u>	firesaber	460.9	<u>1.00</u>	1	firesaber	140.9	<u>1.00</u>	firesaber	150.8	<u>1.00</u>
2	kyber1024	440.6	1.03	kyber1024	503.8	1.09	2	kyber1024	175.4	1.24	kyber1024	198.4	1.31

Encapsulation and Decapsulation 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

neon Cortex-A72							neon Apple M1						
Rank	E	<i>kc</i>	↑	D	<i>kc</i>	↑	Rank	E	<i>kc</i>	↑	D	<i>kc</i>	↑
1	ntru-hrss701	93.6	<u>1.00</u>	kyber512	94.1	<u>1.00</u>	1	ntru-hrss701	22.7	<u>1.00</u>	kyber512	29.4	<u>1.00</u>
2	kyber512	95.3	1.02	lightsaber	131.2	1.39	2	kyber512	32.5	1.43	lightsaber	35.3	1.20
3	lightsaber	130.1	1.39	ntru-hps677	205.8	2.19	3	lightsaber	37.2	1.63	ntru-hps677	54.5	1.85
4	ntru-hps677	181.7	1.94	ntru-hrss701	262.9	2.79	4	ntru-hps677	60.1	2.64	ntru-hrss701	60.7	2.06
1	kyber768	151.0	<u>1.00</u>	kyber768	149.8	<u>1.00</u>	1	kyber768	49.2	<u>1.00</u>	kyber768	45.7	<u>1.00</u>
2	saber	213.6	1.41	saber	215.4	1.44	2	saber	59.9	1.22	saber	58.0	1.27
3	ntru-hps821	232.6	1.54	ntru-hps821	274.5	1.83	3	ntru-hps821	75.7	1.54	ntru-hps821	69.0	1.51
1	kyber1024	223.8	<u>1.00</u>	kyber1024	220.7	<u>1.00</u>	1	kyber1024	71.6	<u>1.00</u>	kyber1024	67.1	<u>1.00</u>
2	firesaber	321.6	1.44	firesaber	329.6	1.49	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						Rank	neon Cortex-A72					
	E	kc	↑	D	kc	↑		E	kc	↑	D	kc	↑
1	lightsaber	154.8	<u>1.00</u>	lightsaber	165.9	<u>1.00</u>	1	ntru-hrss701	93.6	<u>1.00</u>	kyber512	94.1	<u>1.00</u>
2	kyber512	184.5	1.19	kyber512	223.4	1.35	2	kyber512	95.3	1.02	lightsaber	131.2	1.39
3	ntru-hrss701	458.7	2.96	ntru-hps677	1,346.7	8.12	3	lightsaber	130.1	1.39	ntru-hps677	205.8	2.19
4	ntru-hps677	570.8	3.69	ntru-hrss701	1,353.4	8.16	4	ntru-hps677	181.7	1.94	ntru-hrss701	262.9	2.79
1	saber	273.4	<u>1.00</u>	saber	294.5	<u>1.00</u>	1	kyber768	151.0	<u>1.00</u>	kyber768	149.8	<u>1.00</u>
2	kyber768	298.9	1.09	kyber768	349.1	1.19	2	saber	213.6	1.41	saber	215.4	1.44
3	ntru-hps821	748.1	2.74	ntru-hps821	1,830.0	6.21	3	ntru-hps821	232.6	1.54	ntru-hps821	274.5	1.83
1	firesaber	427.3	<u>1.00</u>	firesaber	460.9	<u>1.00</u>	1	kyber1024	223.8	<u>1.00</u>	kyber1024	220.7	<u>1.00</u>
2	kyber1024	440.6	1.03	kyber1024	503.8	1.09	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 ( <i>kilocycles</i> )			Level 3 ( <i>kilocycles</i> )		
	ref	neon	ref/neon	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 ( <i>kc</i> )		neon ( <i>kc</i> )		ref/neon	
	<b>E</b>	<b>D</b>	<b>E</b>	<b>D</b>	<b>E</b>	<b>D</b>
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	<i>kc</i>	↑	D	<i>kc</i>	↑
1	ntru-hrss701	20.4	<u>1.00</u>	kyber512	22.5	<u>1.00</u>
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	<u>1.00</u>	kyber768	35.2	<u>1.00</u>
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	<u>1.00</u>	kyber1024	53.1	<u>1.00</u>
2	firesaber	103.3	1.64	firesaber	103.7	1.95

For **Decapsulation**, the rankings across all security levels are:

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

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

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

For **Encapsulation**, at level 5, the ranking is:

1. CRYSTALS-Kyber, 2. Saber



# NEON vs. AVX2 in Cycles

Algorithm	neon ( <i>kc</i> )		AVX2 ( <i>kc</i> )		AVX2/neon	
	E	D	E	D	E	D
lightsaber	37.2	35.3	41.9	42.2	<u>1.13</u>	<u>1.19</u>
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	<u>1.18</u>	<u>1.22</u>
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	<u>1.18</u>	<u>1.20</u>
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

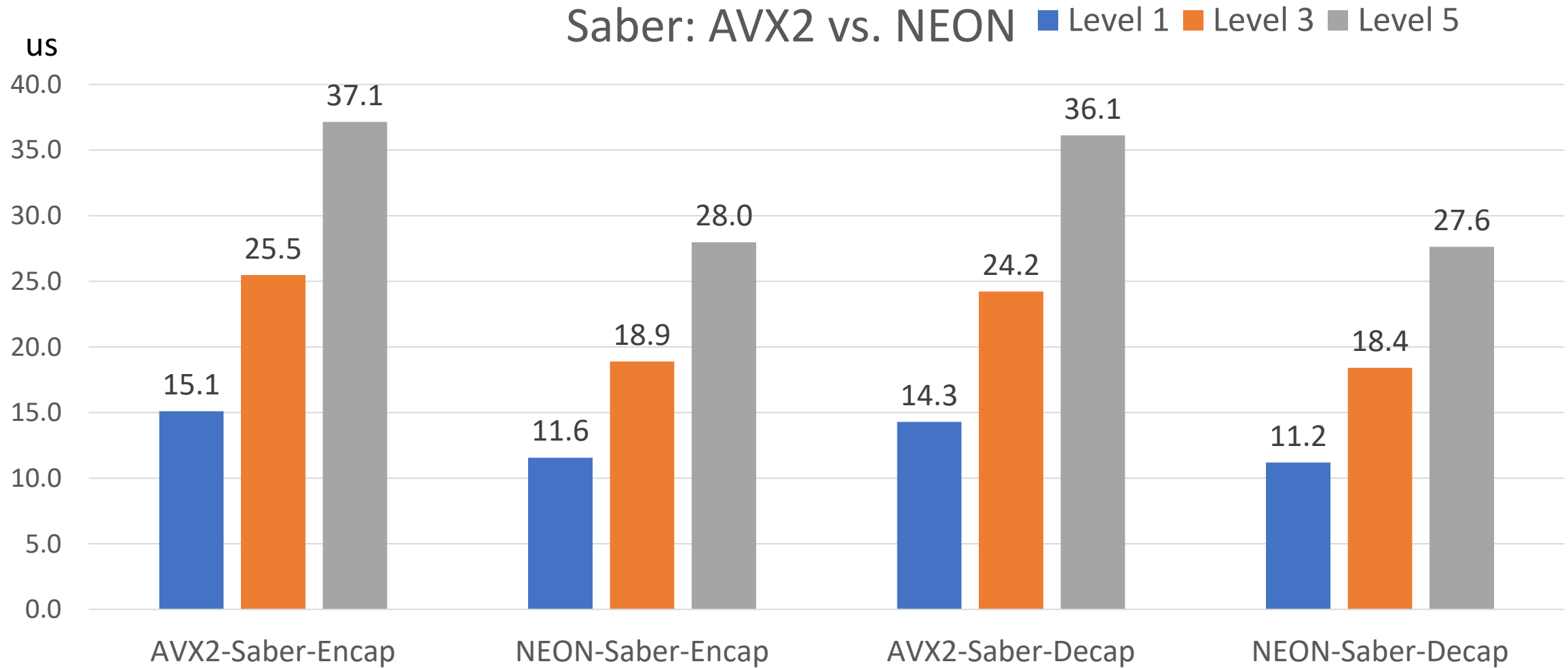
## Apple M1 @ 3.2 GHz versus Intel Core i7-8750H 4.1 GHz

Apple M1 Core i7-8750H	ref ( <i>kc</i> )		neon ( <i>kc</i> )		AVX2 ( <i>kc</i> )		ref/neon		AVX2/neon	
	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

Saber: AVX2 vs. NEON



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.
- New SQDMULH instruction improved the speed of NTT implementation. NTT is faster than Toom-Cook implementation in Saber.

# 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](https://github.com/GMUCERG/PQC_NEON)