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

Cryptographic Engineering Research Group @ George Mason University

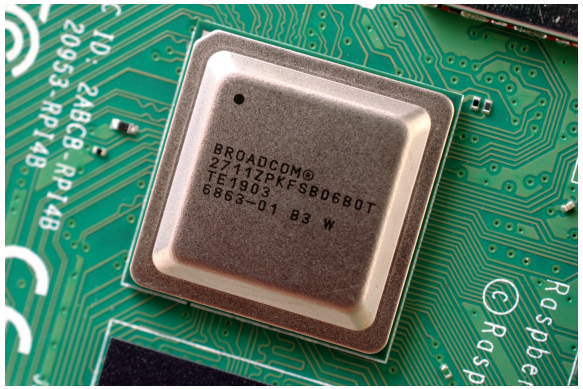
Duc Tri Nguyen and Kris Gaj

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).

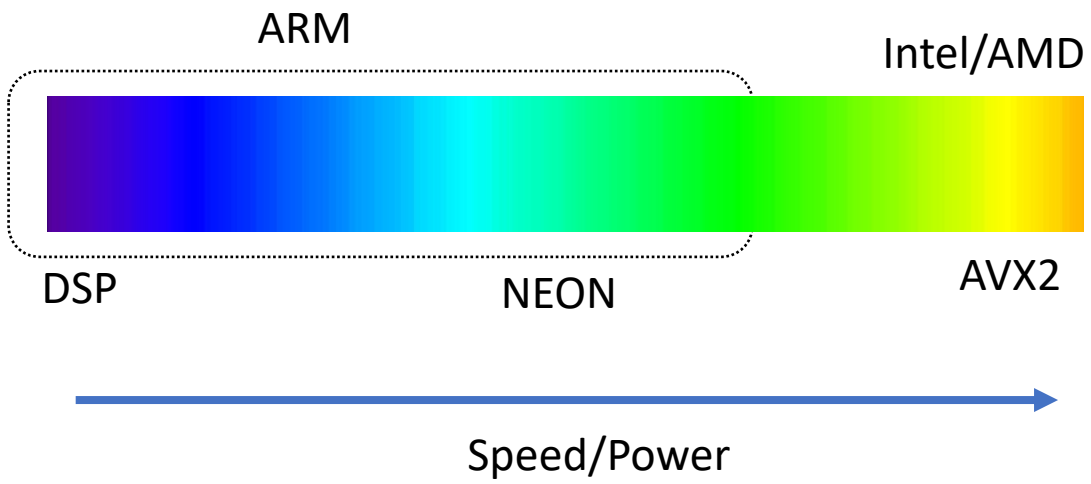
Apple M1:
part of new MacBook Air, MacBook Pro, Mac Mini, iMac, and iPad Pro



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

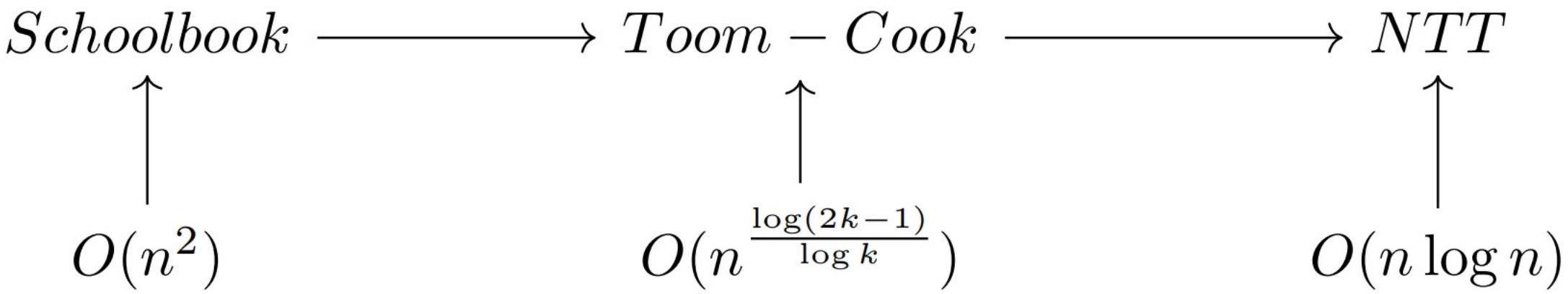
Introduction

- Most software implementations of PQC candidates on:
  - Intel/AMD (w/ AVX2 extension)
  - Cortex-M4 (w/ DSP extension)[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

Polynomial Multiplication



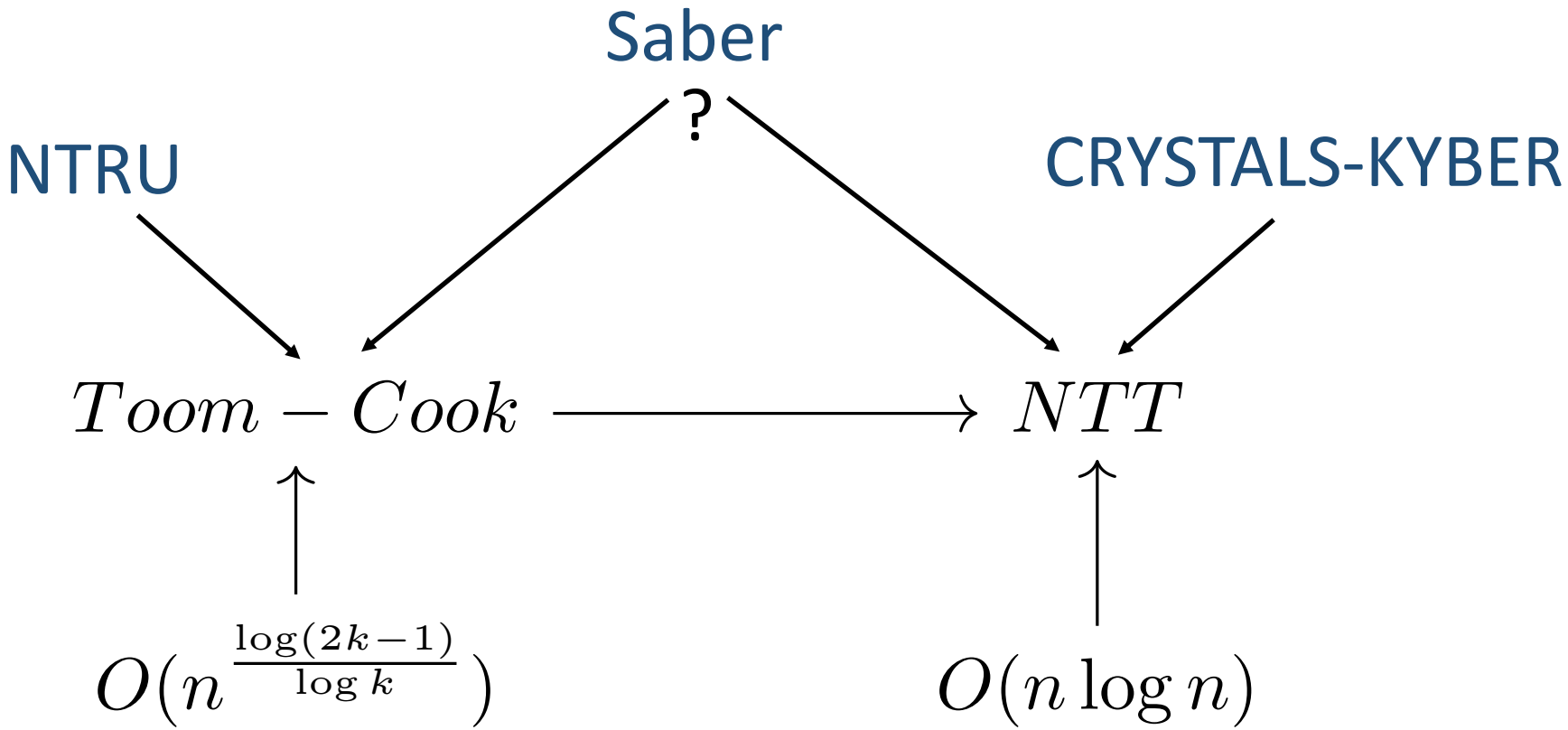
Typically:

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

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

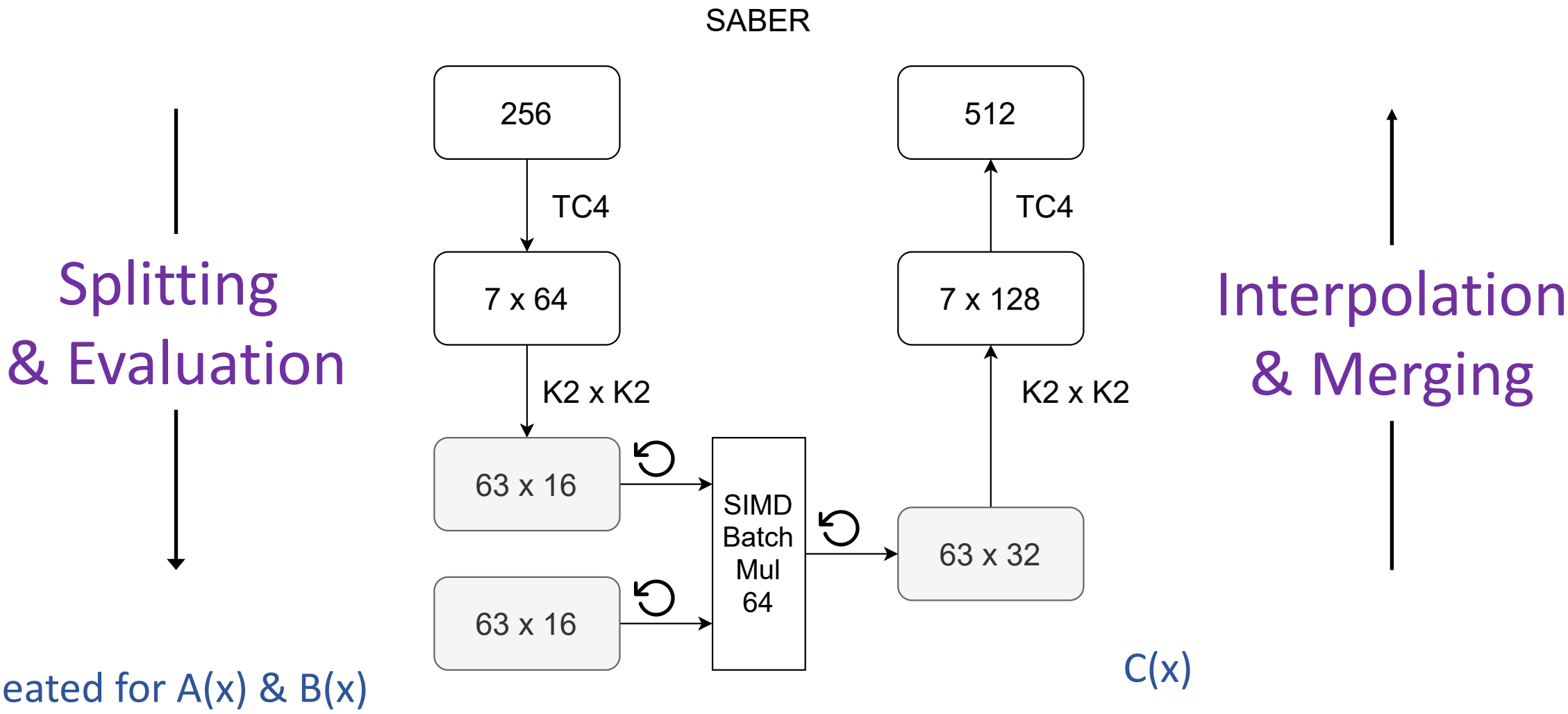
k=4: Toom-4 : $O(n^{1.40})$

Optimal Choice of Algorithms



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

Toom-Cook Implementation: Saber



Schoolbook 16 x 16: Pointwise Multiplication

Number Theoretic Transform

Complete NTT

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

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

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: $A = B * (CR) \mod q$

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

Improved NTT implementation

Algorithm 3: Vectorized multiplication modulo a 16-bit q

Input: B, C, $R = 2^{16}$, $B_{qinv} = B * q^{-1} \mod 2^{16}$
Output: $A = B * (CR) \mod q$

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

- 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 Mulmods implementation uses the **same number of 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

Toom-Cook vs. Improved 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

We select NTT for the implementation of Saber.

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

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	kyber512	88.1	1.00	kyber512	85.1	1.00
2	ntru-hrss701	93.6	1.06	lightsaber	112.3	1.32
3	lightsaber	112.8	1.28	ntru-hps677	205.8	2.42
4	ntru-hps677	181.7	2.06	ntru-hrss701	262.9	3.09
1	kyber768	140.1	1.00	kyber768	136.2	1.00
2	saber	182.0	1.30	saber	185.8	1.36
3	ntru-hps821	232.5	1.66	ntru-hps821	274.5	2.02
1	kyber1024	214.1	1.00	kyber1024	209.7	1.00
2	firesaber	261.2	1.22	firesaber	273.1	1.30

Rank	neon Apple M1 E	kc	↑	D	kc	↑
1	ntru-hrss701	22.8	1.00	kyber512	27.1	1.00
2	kyber512	30.9	1.36	lightsaber	34.1	1.26
3	lightsaber	35.9	1.58	ntru-hps677	54.6	2.02
4	ntru-hps677	60.1	2.64	ntru-hrss701	60.8	2.25
1	kyber768	46.7	1.00	kyber768	41.6	1.00
2	saber	55.9	1.20	saber	54.1	1.30
3	ntru-hps821	75.7	1.62	ntru-hps821	69.0	1.66
1	kyber1024	68.2	1.00	kyber1024	63.5	1.00
2	firesaber	82.8	1.21	firesaber	82.0	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 Apple M1						Rank	neon Apple M1					
	E	*kc*	↑	**D**	*kc*	↑		**E**	*kc*	↑	**D**	*kc*	↑
1	lightsaber	50.9	1.00	lightsaber	54.9	1.00	1	ntru-hrss701	22.8	1.00	kyber512	27.1	1.00
2	kyber512	75.7	1.49	kyber512	89.5	1.63	2	kyber512	30.9	1.36	lightsaber	34.1	1.26
3	ntru-hrss701	152.4	3.00	ntru-hps677	430.4	7.84	3	lightsaber	35.9	1.58	ntru-hps677	54.6	2.02
4	ntru-hps677	183.1	3.60	ntru-hrss701	439.9	8.01	4	ntru-hps677	60.1	2.64	ntru-hrss701	60.8	2.25
1	saber	90.4	1.00	saber	96.2	1.00	1	kyber768	46.7	1.00	kyber768	41.6	1.00
2	kyber768	119.8	1.32	kyber768	137.8	1.43	2	saber	55.9	1.20	saber	54.1	1.30
3	ntru-hps821	245.3	2.71	ntru-hps821	586.5	6.10	3	ntru-hps821	75.7	1.62	ntru-hps821	69.0	1.66
1	firesaber	140.9	1.00	firesaber	150.8	1.00	1	kyber1024	68.2	1.00	kyber1024	63.5	1.00
2	kyber1024	175.4	1.24	kyber1024	198.4	1.31	2	firesaber	82.8	1.21	firesaber	82.0	1.29

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

Apple M1 3200 MHz	Level 1 *(kilocycles)*			Level 3 *(kilocycles)*		
	ref	neon	ref/neon	ref	neon	ref/neon
Saber: Toom-Cook | NTT						
InnerProd	8.0 | 3.1	3.2 | 2.52	2.47	11.8 | 4.3	4.4 | 2.72	2.66
MatrixVectorMul	16.0 | 6.6	5.3 | 2.41	3.00	35.5 | 14.0	9.9 | 2.53	3.56
Kyber						
VectorVectorMul	17.5	1.1	15.9	23.4	1.5	15.5
MatrixVectorMul	27.1	1.7	16.2	46.4	2.9	15.9

Apple M1: NEON vs. Baseline Speed-Up

Algorithm	ref (kc)		neon (kc)		ref/neon	
	E	**D**	**E**	**D**	**E**	**D**
lightsaber	50.9	54.9	35.9	34.1	1.41	1.61
kyber512	75.7	89.5	30.9	27.1	2.45	3.31
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	55.9	54.1	1.62	1.78
kyber768	119.8	137.8	46.7	41.6	2.57	3.32
ntru-hps821	245.3	586.5	75.7	69.0	3.24	8.49
firesaber	140.9	150.8	82.8	82.0	1.70	1.84
kyber1024	175.4	198.4	68.2	63.5	2.57	3.12

NEON M1 vs. AVX2 in Cycles

Algorithm	neon (*kc*)		AVX2 (*kc*)		AVX2/neon	
	E	**D**	**E**	**D**	**E**	**D**
lightsaber	35.9	34.1	41.9	42.2	1.17	1.23
kyber512	30.9	27.1	28.4	22.6	0.92	0.83
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	55.9	54.1	70.9	70.7	1.27	1.31
kyber768	46.7	41.6	43.4	35.2	0.93	0.85
ntru-hps821	75.7	69.0	29.9	57.3	0.39	0.83
firesaber	82.8	82.0	103.3	103.7	1.25	1.26
kyber1024	68.2	63.5	63.0	53.1	0.92	0.84

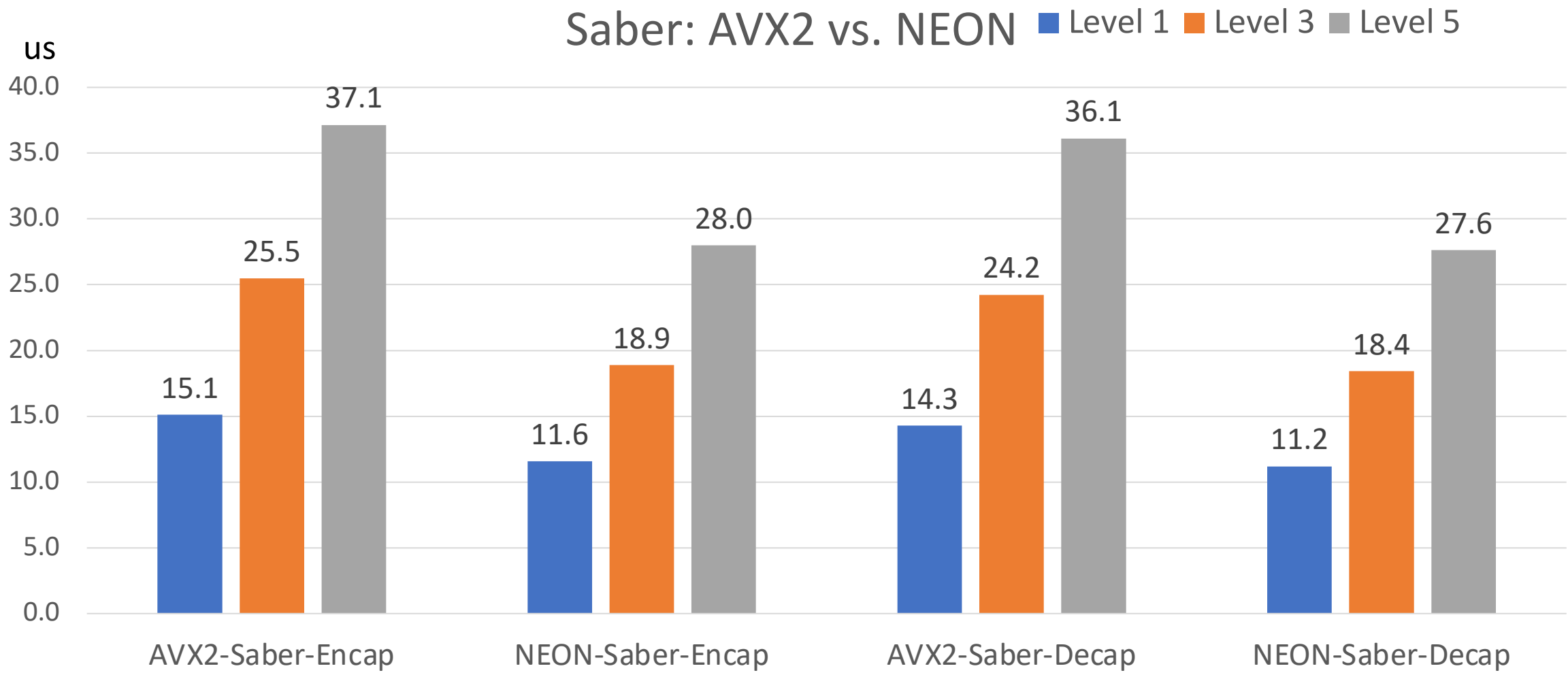
Result for AVX2 AMD EPYC 7742 taken from **supercop-20210125**

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

Apple M1 / Core i7-8750H	ref (*kc*)		neon (*kc*)		AVX2 (*kc*)		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

This slide only, we select Toom-Cook implementation for SABER

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.

- 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 update-to-date source code and results are available at:
https://github.com/GMUCERG/PQC_NEON