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
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 32x128-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
- 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
- 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

\[ \text{Schoolbook} \rightarrow \Toom - \Cook \rightarrow \NTT \]

\[ O(n^2) \rightarrow O\left(n^{\frac{\log(2k-1)}{\log k}}\right) \rightarrow O(n \log n) \]

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

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

3. Pointwise multiplication

\[
\begin{bmatrix}
C(0) \\
C(1) \\
C(-1) \\
C\left(\frac{1}{2}\right) \\
C\left(-\frac{1}{2}\right) \\
C(2) \\
C(\infty)
\end{bmatrix} =
\begin{bmatrix}
A(0) \\
A(1) \\
A(-1) \\
A\left(\frac{1}{2}\right) \\
A\left(-\frac{1}{2}\right) \\
A(2) \\
A(\infty)
\end{bmatrix} \cdot
\begin{bmatrix}
B(0) \\
B(1) \\
B(-1) \\
B\left(\frac{1}{2}\right) \\
B\left(-\frac{1}{2}\right) \\
B(2) \\
B(\infty)
\end{bmatrix}
\]
5 Steps of Toom-4

4. Interpolation

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

where \( C(x) = \sum_{i=0}^{6} \theta_i x^i \)

5. Merging

\[
C(x) = \sum_{i=0}^{2n-1} a_i x^i = \sum_{i=0}^{6} \theta_i x^{i n/4}
\]
Toom-Cook: Splitting & Evaluation

Karatsuba

1 polynomial
N coefficients

3 polynomials
N/2 coefficients each

N \rightarrow K2 \rightarrow 3 \cdot N/2

Toom-3

1 polynomial
N coefficients

5 polynomials
N/3 coefficients each

N \rightarrow TC3 \rightarrow 5 \cdot N/3

Toom-4

1 polynomial
N coefficients

7 polynomials
N/4 coefficients each

N \rightarrow TC4 \rightarrow 7 \cdot N/4
Toom-Cook Implementation: Saber

Splitting & Evaluation

Repeated for A(x) & B(x)

Interpolation & Merging

Schoolbook 16 x 16: Pointwise Multiplication

256
TC4
7 x 64
K2 x K2
63 x 16

512
TC4
7 x 128
K2 x K2
63 x 32
SIMD Batch Mul 64

C(x)
Toom-Cook Implementation: NTRU-HPS821

Splitting & Evaluation

Interpolation & Merging

Repeated for A(x) & B(x)

Schoolbook 16 x 16: Pointwise Multiplication
For multiple layers of Split-Evaluate/Interpolate-Merge:
• unroll these layers to save load/store instructions
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

\[ C(x) = A(x) \times B(x) \]

\[ = NTT^{-1}(C) = NTT^{-1}(A \ast B) = NTT^{-1}(NTT(A) \ast NTT(B)) \]

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

Example of levels

Example of reordering indices between levels
- 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
- Apply to NTT/FFT based submissions
- 16-bit coefficients can reach level 6
- 32-bit coefficients can reach level 5.
NEON dependency chain: \texttt{uzp} and \texttt{smull} (vector unzip and vector multiply)

- Lack of an instruction similar to AVX2 \texttt{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

---

**Algorithm 2**: Vectorized multiplication modulo a 16-bit $q$

- **Input**: $B = (B_L, B_H), C = (C_L, C_H), R = 2^{16}$
- **Output**: $A = B \times (CR) \mod q$

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. $A \leftarrow T_3 - T_0$

---

**Algorithm 15**: Multiplication modulo 16-bit $q$

- **Require**: $-2^{15} \leq a < 2^{15}, \frac{q-1}{2} \leq b \leq \frac{q-1}{2}, b' = bq^{-1} \mod 2^{16}$
- **Ensure**: $r = 2^{16}ab \pmod q$

1. $t_1 \leftarrow \left\lfloor \frac{ab}{2^{16}} \right\rfloor$
2. $t_0 \leftarrow ab' \mod 2^{16}$
3. $t_0 \leftarrow \left\lfloor \frac{2^{16}}{t_0} \right\rfloor$
4. $r \leftarrow (t_1 - t_0) \mod 2^{16}$
Toom-Cook vs. NTT for Saber

All values in cycles

<table>
<thead>
<tr>
<th></th>
<th>Apple M1 3.2 Ghz</th>
<th></th>
<th>Cortex-A72 1.5 Ghz</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Encap</td>
<td>Decap</td>
<td>Encap</td>
<td>Decap</td>
</tr>
<tr>
<td></td>
<td>Toom</td>
<td>NTT</td>
<td>Toom/NTT</td>
<td>Toom</td>
</tr>
<tr>
<td>lightsaber</td>
<td>37,187</td>
<td>43,565</td>
<td>85%</td>
<td>35,318</td>
</tr>
<tr>
<td>saber</td>
<td>59,838</td>
<td>68,867</td>
<td>87%</td>
<td>57,955</td>
</tr>
<tr>
<td>firesaber</td>
<td>87,899</td>
<td>102,206</td>
<td>86%</td>
<td>86,724</td>
</tr>
</tbody>
</table>

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
- 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

<table>
<thead>
<tr>
<th></th>
<th>Apple M1</th>
<th>Encap</th>
<th>Decap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.2 Ghz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.2 Ghz</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lightsaber</td>
<td>37,187</td>
<td>35,949</td>
<td>103%</td>
</tr>
<tr>
<td>saber</td>
<td>59,838</td>
<td>55,892</td>
<td>107%</td>
</tr>
<tr>
<td>firesaber</td>
<td>87,899</td>
<td>82,776</td>
<td>106%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Cortex-A72</th>
<th>Encap</th>
<th>Decap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.5 Ghz</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lightsaber</td>
<td>130,097</td>
<td>116,105</td>
<td>112%</td>
</tr>
<tr>
<td>saber</td>
<td>213,574</td>
<td>183,230</td>
<td>116%</td>
</tr>
<tr>
<td>firesaber</td>
<td>321,637</td>
<td>265,626</td>
<td>121%</td>
</tr>
</tbody>
</table>

**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

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

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


<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](https://github.com/dougallj)
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

### 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**

---

### Table: NEON Implementation Comparison

<table>
<thead>
<tr>
<th>Rank</th>
<th>E</th>
<th>kc</th>
<th>↑</th>
<th>D</th>
<th>kc</th>
<th>↑</th>
<th>Rank</th>
<th>E</th>
<th>kc</th>
<th>↑</th>
<th>D</th>
<th>kc</th>
<th>↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ntru-hrss701</td>
<td>93.6</td>
<td>1.00</td>
<td>kyber512</td>
<td>94.1</td>
<td>1.00</td>
<td>1</td>
<td>ntru-hrss701</td>
<td>22.7</td>
<td>1.00</td>
<td>kyber512</td>
<td>29.4</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>kyber512</td>
<td>95.3</td>
<td>1.02</td>
<td>lightsaber</td>
<td>131.2</td>
<td>1.39</td>
<td>2</td>
<td>kyber512</td>
<td>32.5</td>
<td>1.43</td>
<td>lightsaber</td>
<td>35.3</td>
<td>1.20</td>
</tr>
<tr>
<td>3</td>
<td>lightsaber</td>
<td>130.1</td>
<td>1.39</td>
<td>ntru-hps677</td>
<td>205.8</td>
<td>2.19</td>
<td>3</td>
<td>lightsaber</td>
<td>37.2</td>
<td>1.63</td>
<td>ntru-hps677</td>
<td>54.5</td>
<td>1.85</td>
</tr>
<tr>
<td>4</td>
<td>ntru-hps677</td>
<td>181.7</td>
<td>1.94</td>
<td>ntru-hrss701</td>
<td>262.9</td>
<td>2.79</td>
<td>4</td>
<td>ntru-hps677</td>
<td>60.1</td>
<td>2.64</td>
<td>ntru-hrss701</td>
<td>60.7</td>
<td>2.06</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rank</th>
<th>E</th>
<th>kc</th>
<th>↑</th>
<th>D</th>
<th>kc</th>
<th>↑</th>
<th>Rank</th>
<th>E</th>
<th>kc</th>
<th>↑</th>
<th>D</th>
<th>kc</th>
<th>↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>kyber768</td>
<td>151.0</td>
<td>1.00</td>
<td>kyber768</td>
<td>149.8</td>
<td>1.00</td>
<td>1</td>
<td>kyber768</td>
<td>49.2</td>
<td>1.00</td>
<td>kyber768</td>
<td>45.7</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>saber</td>
<td>213.6</td>
<td>1.41</td>
<td>saber</td>
<td>215.4</td>
<td>1.44</td>
<td>2</td>
<td>saber</td>
<td>59.9</td>
<td>1.22</td>
<td>saber</td>
<td>58.0</td>
<td>1.27</td>
</tr>
<tr>
<td>3</td>
<td>ntru-hps821</td>
<td>232.6</td>
<td>1.54</td>
<td>ntru-hps821</td>
<td>274.5</td>
<td>1.83</td>
<td>3</td>
<td>ntru-hps821</td>
<td>75.7</td>
<td>1.54</td>
<td>ntru-hps821</td>
<td>69.0</td>
<td>1.51</td>
</tr>
<tr>
<td>1</td>
<td>kyber1024</td>
<td>223.8</td>
<td>1.00</td>
<td>kyber1024</td>
<td>220.7</td>
<td>1.00</td>
<td>1</td>
<td>kyber1024</td>
<td>71.6</td>
<td>1.00</td>
<td>kyber1024</td>
<td>67.1</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>firesaber</td>
<td>321.6</td>
<td>1.44</td>
<td>firesaber</td>
<td>329.6</td>
<td>1.49</td>
<td>2</td>
<td>firesaber</td>
<td>87.9</td>
<td>1.23</td>
<td>firesaber</td>
<td>86.7</td>
<td>1.29</td>
</tr>
</tbody>
</table>
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
<table>
<thead>
<tr>
<th></th>
<th>Cortex-A72 1500 MHz</th>
<th>Level 1 (kilocycles)</th>
<th>Level 3 (kilocycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ref</td>
<td>neon</td>
<td>ref/neon</td>
</tr>
<tr>
<td>InnerProd</td>
<td>27.7</td>
<td>18.1</td>
<td>22.5</td>
</tr>
<tr>
<td>MatrixVectorMul</td>
<td>55.2</td>
<td>40.2</td>
<td>37.0</td>
</tr>
<tr>
<td>VectorVectorMul</td>
<td>44.4</td>
<td>7.1</td>
<td>6.3</td>
</tr>
<tr>
<td>MatrixVectorMul</td>
<td>68.1</td>
<td>10.7</td>
<td>6.4</td>
</tr>
</tbody>
</table>
### NEON vs. Baseline Speed-Up

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>ref (kc)</th>
<th>neon (kc)</th>
<th>ref/neon</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>lightsaber</td>
<td>50.9</td>
<td>54.9</td>
<td>37.2</td>
</tr>
<tr>
<td>kyber512</td>
<td>75.7</td>
<td>89.5</td>
<td>32.6</td>
</tr>
<tr>
<td>ntru-hps677</td>
<td>183.1</td>
<td>430.4</td>
<td>60.1</td>
</tr>
<tr>
<td>ntru-hrss701</td>
<td>152.4</td>
<td>439.9</td>
<td>22.8</td>
</tr>
<tr>
<td>saber</td>
<td>90.4</td>
<td>96.2</td>
<td>59.9</td>
</tr>
<tr>
<td>kyber768</td>
<td>119.8</td>
<td>137.8</td>
<td>49.2</td>
</tr>
<tr>
<td>ntru-hps821</td>
<td>245.3</td>
<td>586.5</td>
<td>75.7</td>
</tr>
<tr>
<td>firesaber</td>
<td>140.9</td>
<td>150.8</td>
<td>87.9</td>
</tr>
<tr>
<td>kyber1024</td>
<td>175.4</td>
<td>198.4</td>
<td>71.6</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Rank</th>
<th>E</th>
<th>kc</th>
<th>↑</th>
<th>D</th>
<th>kc</th>
<th>↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ntru-hrss701</td>
<td>20.4</td>
<td>1.00</td>
<td>kyber512</td>
<td>22.5</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>ntru-hps677</td>
<td>25.9</td>
<td>1.27</td>
<td>lightsaber</td>
<td>42.1</td>
<td>1.87</td>
</tr>
<tr>
<td>3</td>
<td>kyber512</td>
<td>28.3</td>
<td>1.39</td>
<td>ntru-hps677</td>
<td>45.7</td>
<td>2.03</td>
</tr>
<tr>
<td>4</td>
<td>lightsaber</td>
<td>41.9</td>
<td>2.05</td>
<td>ntru-hrss701</td>
<td>47.6</td>
<td>2.11</td>
</tr>
<tr>
<td></td>
<td>ntru-hps821</td>
<td>29.9</td>
<td>1.00</td>
<td>kyber768</td>
<td>35.2</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>kyber768</td>
<td>43.4</td>
<td>1.45</td>
<td>saber</td>
<td>57.3</td>
<td>1.63</td>
</tr>
<tr>
<td>3</td>
<td>saber</td>
<td>70.9</td>
<td>2.37</td>
<td>ntru-hps821</td>
<td>70.7</td>
<td>2.01</td>
</tr>
<tr>
<td>1</td>
<td>kyber1024</td>
<td>63.0</td>
<td>1.00</td>
<td>kyber1024</td>
<td>53.1</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>firesaber</td>
<td>103.3</td>
<td>1.64</td>
<td>firesaber</td>
<td>103.7</td>
<td>1.95</td>
</tr>
</tbody>
</table>
### NEON vs. AVX2 in Cycles

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>neon ((kc))</th>
<th>AVX2 ((kc))</th>
<th>AVX2/neon</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>lightsaber</td>
<td>37.2</td>
<td>35.3</td>
<td>41.9</td>
</tr>
<tr>
<td>kyber512</td>
<td>32.6</td>
<td>29.4</td>
<td>28.4</td>
</tr>
<tr>
<td>ntru-hps677</td>
<td>60.1</td>
<td>54.6</td>
<td>26.0</td>
</tr>
<tr>
<td>ntru-hrss701</td>
<td>22.8</td>
<td>60.8</td>
<td>20.4</td>
</tr>
<tr>
<td>saber</td>
<td>59.9</td>
<td>58.0</td>
<td>70.9</td>
</tr>
<tr>
<td>kyber768</td>
<td>49.2</td>
<td>45.7</td>
<td>43.4</td>
</tr>
<tr>
<td>ntru-hps821</td>
<td>75.7</td>
<td>69.0</td>
<td>29.9</td>
</tr>
<tr>
<td>firesaber</td>
<td>87.9</td>
<td>86.7</td>
<td>103.3</td>
</tr>
<tr>
<td>kyber1024</td>
<td>71.6</td>
<td>67.1</td>
<td>63.0</td>
</tr>
</tbody>
</table>

Result for AVX2 AMD EPYC 7742 taken from supercop-20210125
<table>
<thead>
<tr>
<th>Apple M1 Core i7-8750H</th>
<th>ref ((kc))</th>
<th>neon ((kc))</th>
<th>AVX2 ((kc))</th>
<th>ref/neon</th>
<th>AVX2/neon</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E</td>
<td>D</td>
<td>E</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>NTRU-HPS677</td>
<td>183.1</td>
<td>430.4</td>
<td>60.1</td>
<td>54.6</td>
<td>47.6</td>
</tr>
<tr>
<td>NTRU-HRSS701</td>
<td>152.4</td>
<td>439.9</td>
<td>22.8</td>
<td>60.8</td>
<td>28.8</td>
</tr>
<tr>
<td>LIGHTSABER</td>
<td>50.9</td>
<td>54.9</td>
<td><strong>37.2</strong></td>
<td><strong>35.3</strong></td>
<td><strong>35.1</strong></td>
</tr>
<tr>
<td>KYBER512</td>
<td>75.7</td>
<td>89.5</td>
<td>32.6</td>
<td>29.4</td>
<td>23.2</td>
</tr>
<tr>
<td>NTRU-HPS821</td>
<td>245.3</td>
<td>586.5</td>
<td>75.7</td>
<td>69.0</td>
<td>56.1</td>
</tr>
<tr>
<td>SABER</td>
<td>90.4</td>
<td>96.2</td>
<td><strong>59.9</strong></td>
<td><strong>58.0</strong></td>
<td><strong>54.3</strong></td>
</tr>
<tr>
<td>KYBER768</td>
<td>119.8</td>
<td>137.8</td>
<td>49.2</td>
<td>45.7</td>
<td>33.9</td>
</tr>
<tr>
<td>FIRESABER</td>
<td>140.9</td>
<td>150.8</td>
<td><strong>87.9</strong></td>
<td><strong>86.7</strong></td>
<td><strong>78.9</strong></td>
</tr>
<tr>
<td>KYBER1024</td>
<td>175.4</td>
<td>198.4</td>
<td>71.6</td>
<td>67.1</td>
<td>45.2</td>
</tr>
</tbody>
</table>

Frequency Scaling Effect

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