Highly Efficient Architecture of NewHope-NIST on FPGA using Low-Complexity NTT/INTT

Neng Zhang, Bohan Yang, Chen Chen, Shouyi Yin, Shaojun Wei and Leibo Liu

Institute of Microelectronics, Tsinghua University, Beijing, China
Outline

- 1. Introduction
- 2. Low-Complexity NTT/INTT
- 3. Hardware Architecture
- 4. Implementation Results
1 Introduction

- NewHope: a PQC algorithm for key encapsulation mechanism (KEM)

  - NewHope-USENIX
  - NewHope-Simple
  - NewHope-NIST

- A candidate in the 2\textsuperscript{nd} round of NIST PQC standardization process, but not in the 3\textsuperscript{rd} round

- Low-complexity NTT/INTT can be utilized by other algorithms.
1 Introduction

- **Main mathematical objects of NewHope**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q$</td>
<td>12289</td>
<td></td>
</tr>
<tr>
<td>$\omega_N$</td>
<td></td>
<td>Primitive $N$-th root of unit over $\mathbb{Z}_q$</td>
</tr>
<tr>
<td>$N$</td>
<td>1024 or 512</td>
<td>Square root of $\omega_N$</td>
</tr>
</tbody>
</table>

- **Encryption-based KEM**

<table>
<thead>
<tr>
<th>Phase</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Generation</td>
<td>2 NTTs</td>
</tr>
<tr>
<td>Encryption</td>
<td>2 NTTs, 1 INTT</td>
</tr>
<tr>
<td>Decryption</td>
<td>1 INTT</td>
</tr>
</tbody>
</table>
1 Introduction

Multiplication over the ring $\mathbb{Z}_q[x]/f(x)$

- $f(x)$ is arbitrary
  - Convolution theory
  - $q \equiv 1 \pmod{N}$

- $f(x) = x^N + 1$
  - Negative Wrapped Convolution (NWC)
  - $q \equiv 1 \pmod{2N}$

Diagram:

- $a$ and $b$ are input signals.
- Zero-padding is applied to both inputs.
- 2N-point FFTs are performed on both inputs.
- The results are multiplied.
- 2N-point FFTs are applied again to the result.
- Modulo $f(x)$ is performed.
- The final result is obtained.

Institute of Microelectronics, Tsinghua University.
1 Introduction

- Why do we need low-complexity?

- Low area
- High speed

Area vs. Speed:
- BRAM
- DSP
- Register
- LUT

- Frequency
- Clock cycles

Low-complexity

Institute of Microelectronics, Tsinghua University.
2.1 Low-Complexity NTT

- Cost of the pre-processing is considerable:
  \[(N/2) \log N + N\]
  
  \[\uparrow\quad \uparrow\]
  FFT  pre-processing

- Low-Complexity NTT:
  ➢ A low-complexity NTT with twiddle factors computed on-the-fly [1].
  ➢ Merge the pre-processing into the DIT FFT with twiddle factors pre-computed.

2.1 Low-Complexity NTT

Derivation of the low-complexity NTT

- Inspired by the strategy of the Cooley-Turkey FFT
- Follow the divide-and-conquer method of FFT that divides in time domain (DIT)

First, the pre-processing and the FFT are written together as a summation of N items

\[ \hat{a}_i = \sum_{j=0}^{N-1} a_j \gamma_{2N}^j \omega_N^{ij} \mod q \]

Second, the summation is split into two groups according to parity of the index of \( a \)

\[ \hat{a}_i = \sum_{j=0}^{N/2-1} a_{2j} \omega_N^{2ij} \gamma_{2N}^{2j} + \sum_{j=0}^{N/2-1} a_{2j+1} \omega_N^{i(2j+1)} \gamma_{2N}^{2j+1} \mod q \]
2.1 Low-Complexity NTT

Derivation of the low-complexity NTT

➢ Third, the equation is grouped into two parts according to the size of index \(i\).

\[
\hat{a}_i^{(0)} = \sum_{j=0}^{N/2-1} a_{2j} \omega_{N/2}^j \gamma_N^j \mod q,
\hat{a}_i^{(1)} = \sum_{j=0}^{N/2-1} a_{2j+1} \omega_{N/2}^j \gamma_N^j \mod q.
\]

\[
\hat{a}_i = \hat{a}_i^{(0)} + \omega_N^i \gamma_2^2 \hat{a}_i^{(1)} \mod q
\]
\[
\hat{a}_{i+N/2} = \hat{a}_i^{(0)} - \omega_N^i \gamma_2^2 \hat{a}_i^{(1)} \mod q
\]

\(\hat{a}_i^{(0)}\) and \(\hat{a}_i^{(1)}\) are N/2-point NTTs of \(a_{2j}\) and \(a_{2j+1}\)

➢ In this way, N-point NTT can be resolved with two N/2-point NTTs
2.1 Low-Complexity NTT

Dataflow of a 8-point low-complexity NTT

\[ \omega_{2m} Y_{2m} \]

Butterfly of low-complexity NTT

\[ a = a + b \omega_m^1 Y_{2m} \mod q \]

\[ b = a - b \omega_m^1 Y_{2m} \mod q \]

\[ \omega_{2m}^j \gamma_{2m} \equiv \gamma_{2m}^{2j+1} \]

\[ \equiv \gamma_{2N}^{(2j+1)N/m} \mod q \]
2.1 Low-Complexity NTT

Algorithm 4 Low-Complexity NTT Algorithm without Pre-processing

Let the vectors \( \mathbf{a} \) and \( \mathbf{A} \) denote \((a_0, a_1, ..., a_{N-1})\) and \((A_0, A_1, ..., A_{N-1})\), respectively, where \(a_i \in \mathbb{Z}_q\), \(A_i \in \mathbb{Z}_q\), \(i = 1, 2, ..., N - 1\). Let \(\omega_N\) be a primitive \(N\)-th root of unity in \(\mathbb{Z}_q\) and let \(\gamma_{2N} = \sqrt{\omega_N}\).

Input: \(\mathbf{a}, N, q; \gamma_{2N}^i, i = 0, 1, ..., N - 1\).
Output: \(\mathbf{A} = \text{NTT}(\mathbf{a})\)

1: \(\mathbf{A} \leftarrow \text{scramble}(\mathbf{a})\)
2: for \(s = 1\) to \(\log_2 N\) do
3: \(m \leftarrow 2^s\)
4: for \(j = 0\) to \(m/2 - 1\) do
5: \(\omega = \gamma_{2N}^{(2j+1)N/m}\)
6: for \(k = 0\) to \(N/m - 1\) do
7: \(u = A_{km+j}\)
8: \(t = \omega \cdot A_{km+j+m/2} \mod q\)
9: \(A_{km+j} = (u + t) \mod q\)
10: \(A_{km+j+m/2} = (u - t) \mod q\)
11: end for
12: end for
13: end for
14: return \(\mathbf{A}\)

In classic FFT:
\[
\omega = \omega_N^{jN/m}
\]

Computational complexity:

\((N/2) \log N + N \rightarrow (N/2) \log N\)

No additional timing cost;
No additional hardware resources cost
2.2 Low-Complexity INTT

- Cost of the post-processing is greater than pre-processing
  \[(N/2) \log N + 2N\]
  \[
  \uparrow \quad \text{FFT} \quad \uparrow \quad \text{post-processing}
  \]

- Low-Complexity INTT
  - [1] merges the scaling of \(\lambda_{2N}^{-i}\) into the FFT.
  - Further merge the scaling of \(N^{-1}\) into the FFT

2.2 Low-Complexity INTT

- Inspired by the strategy of the Gentleman-Sande FFT
- Follow the divide-and-conquer method of FFT that divides in frequency domain (DIF)

- First, the post-processing and the FFT are written together as a summation of N items

\[
a_i = N^{-1} \gamma_{2N}^{-i} \sum_{j=0}^{N-1} \hat{a}_j \omega_N^{-ij} \mod q
\]

- Second, the summation is split into two groups according to the size of index of \( \hat{a} \)

\[
a_i = N^{-1} \gamma_{2N}^{-i} \left( \sum_{j=0}^{N/2-1} \hat{a}_j \omega_N^{-ij} + \sum_{j=N/2}^{N-1} \hat{a}_j \omega_N^{-ij} \right) \mod q
\]
2.2 Low-Complexity INTT

Derivation of the low-complexity INTT

➢ Third, the equation is grouped into two parts according to the parity of \(i\).

\[
\hat{b}_j^{(0)} = \frac{\hat{a}_j + \hat{a}_{(j+N/2)}}{2} \mod q, \quad \hat{b}_j^{(1)} = \left[\frac{\hat{a}_j - \hat{a}_{(j+N/2)}}{2}\right] \omega_N^{-j} \gamma_2^{-1} \mod q
\]

\[
a_{2i} = \left(\frac{N}{2}\right)^{-1} \gamma_N^{-i} \sum_{j=0}^{N/2-1} \hat{b}_j^{(0)} \omega_N^{-ij} \mod q
\]

\[
a_{2i+1} = \left(\frac{N}{2}\right)^{-1} \gamma_N^{-i} \sum_{j=0}^{N/2-1} \hat{b}_j^{(1)} \omega_N^{-ij} \mod q
\]

➢ In this way, N-point INTT can be resolved with two N/2-point INTTs

\[a_{2i} \text{ and } a_{2i+1} \text{ correspond to N/2-point INTT of } \hat{b}_i^{(0)} \text{ and } \hat{b}_i^{(1)}\]
2.2 Low-Complexity INTT

Dataflow of a 8-point low-complexity INTT

Butterfly of low-complexity INTT

\[ A = \frac{1}{2} (a + b) \mod q \]

\[ B = \frac{1}{2} (a - b) \omega_m^{-j} \gamma_{2m}^{-1} \mod q \]

\[ \omega_m^{-j} \gamma_{2m}^{-1} \equiv \gamma_{2m}^{-(2j+1)} \]

\[ \equiv \gamma_{2N}^{-(2j+1)N/m} \pmod{q} \]
2.2 Low-Complexity INTT

Algorithm 5 Low-Complexity INTT Algorithm without Post-processing

Let the vectors \( a \) and \( A \) denote \((a_0, a_1, ..., a_{N-1})\) and \((A_0, A_1, ..., A_{N-1})\), respectively, where \( a_i \in \mathbb{Z}_q, A_i \in \mathbb{Z}_q, i = 1, 2, ..., N - 1 \). Let \( \omega_N \) be a primitive \( N \)-th root of unity in \( \mathbb{Z}_q \) and \( \gamma_{2N} = \sqrt{\omega_N} \).

Input: \( a, N, q, \gamma_{2N}^{-1} \), where \( 0, 1, ..., N - 1 \).

Output: \( A = INTT(a) \)

1: \( s = \log_2 N \)
2: \( m \leftarrow 2^s \)
3: \( \omega = \gamma_{2N}^{-(2j+1)N/m} \)
4: \( \omega = \omega_N^{-jN/m} \)
5: \( u = A_{km+j} \)
6: \( t = A_{km+j+m/2} \mod q \)
7: \( A_{km+j} = \frac{u + t}{2} \mod q \)
8: \( A_{km+j+m/2} = \frac{u - t}{2} \mod q \)
9: \( u + t \)
10: \( u - t \)
11: \( \) end for
12: \( \) end for
13: \( A \leftarrow scramble(a) \)
14: \( \) return \( A \)

In classic FFT:
\[
\omega = \omega_N^{-jN/m}
\]

Computational complexity:
\[
\frac{N}{2} \log N + 2N \quad \rightarrow \quad \frac{N}{2} \log N
\]

No additional timing cost; slightly modify the butterfly unit.
3 The Hardware Architecture

☐ The architecture of NTT/INTT

☐ Multi-bank memory

Address generator [1]:

\[
\text{BankAddr} = \sum_{i=0}^{\left\lfloor \frac{1}{2} \log_2 N \right\rfloor - 1} \text{addr}[2i + 1 : 2i] \mod 4
\]

\[
\text{NewAddr} = \text{addr} >> 2,
\]

- Log N: Even ✓ Odd ×
- The execution order of the last s-loop is rearranged as:

\[
\begin{align*}
\text{for } j = 0 \text{ to } N/4 - 1
A_j &\leftarrow A_j + \gamma_{2N}^{2j+1} A_{j+N/2} \\
A_{j+N/2} &\leftarrow A_j - \gamma_{2N}^{2j+1} A_{j+N/2} \\
A_{j+N/4} &\leftarrow A_{j+N/4} + \gamma_{2N}^{2j+N/2+1} A_{j+3N/4} \\
A_{j+3N/4} &\leftarrow A_{j+N/4} - \gamma_{2N}^{2j+N/2+1} A_{j+3N/4}
\end{align*}
\]

3 The Hardware Architecture

Compact Butterfly Unit

\[ A = a + b w_m^1 y_{2m}^1 \mod q \]
\[ B = a - b w_m^1 y_{2m}^1 \mod q \]

\[ a \]
\[ \frac{1}{2} \]
\[ A = \frac{1}{2} (a + b) \mod q \]
\[ b \]
\[ \frac{1}{2} \]
\[ B = \frac{1}{2} (a - b) w_m^1 y_{2m}^{-1} \mod q \]

\[ \frac{x}{2} \equiv (2 \left\lfloor \frac{x}{2} \right\rfloor + 1) \frac{q + 1}{2} \equiv \left\lfloor \frac{x}{2} \right\rfloor (q + 1) + \frac{q + 1}{2} \equiv \left\lfloor \frac{x}{2} \right\rfloor + \frac{q + 1}{2} \] (mod q)
3 The Hardware Architecture

- Low-Complexity Modular Multiplication

\[ 2^{14} \equiv 2^{12} - 1 \mod 12289 \]

\[
z \equiv 2^{14}z[27:14] + z[13:0] \\
\]

... 

No additional multiplication;

Time-constant
3 The Hardware Architecture

- The architecture of NewHope-NIST

  - Support: key generation, encryption and decryption
  - Doubled bandwidth matching
  - RAM (R0, R1): two data in an address
3 The Hardware Architecture

- Timing hiding
  - Resource conflict
  - Data dependency

A RAM may be read and write by operations in the same line.

Algorithm 7 Pseudo-code for implementation of NewHope-CPA-PKE Encryption

Input: \( pk, \mu, \text{coin} \).
Output: \( \text{EncodePolynomial}(\hat{\mu}), h \).

1: \( R_2 \leftarrow \text{Sample}(\text{coin}, 0) \)
2: \( R_2 \leftarrow \text{NTT}(R_2); R_0 \leftarrow \text{GenA}(pk[0 : 31]) \)
3: \( R_0 \leftarrow R_0 \odot R_2; R_1 \leftarrow R_2; R_2 \leftarrow \text{Sample}(\text{coin}, 1) \)
4: \( R_0 \leftarrow \text{NTT}(R_0) \)
5: output \( \text{EncodePolynomial}(R_0 + R_2) \); \( R_2 \leftarrow \text{DecodePolynomial}(pk[32 : 7n/4 + 31]) \)
6: \( R_2 \leftarrow R_2 \odot R_1; R_0 \leftarrow \text{Sample}(\text{coin}, 2) \)
7: \( R_2 \leftarrow \text{INTT}(R_2) \)
8: \( R_0 \leftarrow \text{PolyBitRev}(R_2) + R_0; R_1 \leftarrow \text{Encode}(\mu) \)
9: output \( \text{Compress}(R_0 + R_1) \)
4 Implementation Results

Implementation platform

- Xilinx Artix-7 FPGA
- Vivado 2019.1.1

Implementation Results of NTT/INTT
4 Implementation Results

□ Implementation Results of NewHope-NIST

- Ours
- [JGCS19-1]
- [JGCS19-2]
- [buc19b]
- [FSM+19]
Conclusion

- Low-complexity NTT/INTT
  - NTT: no pre-processing
  - INTT: no post-processing

- A highly efficient architecture of NewHope-NIST
  - A clear advantage in both speed and ATP

- Low-complexity NTT/INTT can benefit other NTT-inside algorithms
Thanks!